AN INVESTIGATION OF TURBO CODES OVER MOBILE WIRELESS CHANNELS

CHRISTOPHER PAUL DENNETT

A thesis submitted in partial fulfilment of the requirements of the University of Wolverhampton for the degree of Doctor of Philosophy

October 2006

This work or any part thereof has not previously been presented in any form to the University or to any other body whether for the purposes of assessment, publication or for any other purpose (unless otherwise indicated). Save for any express acknowledgements, references and/or bibliographies cited in the work, I confirm that the intellectual content of the work is the result of my own efforts and of no other person.

Signature ……

Date ……

Acknowledgements

Most importantly, thanks must go to the Chief, Professor Rolando Carrasco, who was incredibly patient, insightful and supportive during the project.

Special thanks to my parents, Yvonne and George, and my sister Nikki, who offered love, support and bucket loads of encouragement when days were dark.

Big thanks to all the researchers I worked with at Staffordshire University and the University of Wolverhampton, especially Jules, Bert, Cam and Martin.

I would also like to thank my colleagues at the University of Wolverhampton, Dr. Kamal Bechkoum and Prof. Andy Sloane, for their help and support through the submission process, and the whole of the Databases and Distributed Systems division for keeping me relatively sane during the final slog.

Finally, I'd like to thank my wonderful wife, Virginie, who has always been there for me, ready to listen and offer encouragement. I couldn't have done it without her.

Chris Dennett

Wolverhampton, October 2006

Abstract

Turbo codes have been the subject of much research in recent years, producing results very close to the theoretical limit set by Shannon. The codes have been successfully implemented in satellite and video conferencing systems and provision has been made in 3rd generation mobile systems. These codes have not been used for short frame systems due to the delay at the decoder. In this thesis, comprehensive comparisons of the two common decoding algorithms are made, with reference to short frames. The effects of increasing memory size of component codes, frame sizes, utilising puncturing and errors in channel estimation are investigated over AWGN and Rayleigh fading channels. The decoder systems are compared for complexity as well as for equal numbers of iterations. Results show that less complex decoder strategies produce good results for voice quality bit error rates.

Investigations are also made into the effects of errors in signal-to-noise ratio estimation at the SOVA turbo decoder, showing this decoding algorithm to be more resilient than Log-MAP decoders in published literature. The decoders are also tested over channels displaying inter-symbol interference. Channels include a time-invariant channel and three ETSI standard time-varying channels simulating indoor, pedestrian and vehicular situations, upgraded for more realistic Doppler effect. To combat these types of channels, a derivative of turbo codes, turbo equalisation is often used. To keep receiver delay to a minimum, decision feedback equalisation is used here. Results show that the combination can produce improvements in decoded results with increasing turbo iterations where ISI is low, but that iterative improvements do not occur under harsh circumstances. The combination produces much superior results compared with codes on their own under even the most extreme circumstances.

Contents

Chapter

1

Introduction

1.1 Introduction and Motivation

The fundamental task of a communications system is to transmit and receive information, be it in voice or data form. In theory, a communications system requires that information be transferred from transmitter to receiver in such a way that the information received is of the same quality as that transmitted. In practice, the presence of distortion in various forms means that achieving this is not always possible, especially where wireless transmission is concerned. To achieve something close to the transmitted information at the receiver, a communications system must guard against the changes that this distortion can cause in the transmitted data. Error control coding is one method available to the communications system designer, arranging and supplementing the data in such a way that errors can be corrected at the receiver.

In 1948, Shannon introduced the concept of channel capacity, describing the limit to the amount of data that could be transmitted across any given channel. Since then, attaining this maximum theoretical channel capacity has been the goal of many mobile communications researchers. To understand the rising need to reach this theoretical limit, consider the modern mobile communications industry. In recent years, the use of mobile communications has grown at an enormous rate, not only in numbers of users, but also in the numbers and complexity of applications. For example, ten years ago, mobile phones were used simply for telephone calls and, to a lesser extent, text messages; now they are used to send photographs and videos and to access the internet. Internationally, the number of users has risen from two hundred million in 1996 to 1.2 billion this year, with an estimated 2 billion by the year 2010 (source: www.umts-forum.org). As with everything in this technological age, users demand faster and higher quality applications. As a result, not only are more and more people using mobile phones, they are also demanding higher data rates.

Multiplexing techniques exist to share a given channel between multiple users, which is an aid to handling the increases in user numbers, with newer techniques such as Code Division Multiple Access (CDMA) permitting 10 to 20 times the number of concurrent users that first generation mobile

systems could allow. However, these systems do not help to increase data rate. There are only two ways to do this. The first, increasing the bandwidth of the channel, clearly reduces the number of available channels and therefore does not solve the problems created by the growth in user numbers. The second method is to improve the coding scheme, such that the available channel is used to its full potential. This not only allows higher data rates but has the added effect of reducing the bandwidth necessary for many applications, allowing more channels for a particular bandwidth range.

In 1993, Berrou *et al* published results for a new coding scheme that were very near to Shannon's limit [BER93]. The scheme, dubbed turbo codes, relies on an iterative soft decoding scheme. It uses two encoders in parallel concatenation, separated by an interleaver to increase the average code weight output prior to transmission. The receiver is based around two component decoders in serial concatenation. Each decoder uses information garnered from the previous decoder as an aid to its error control process. This system can be run repeatedly for one received codeword, returning improved results with each successive iteration.

To understand the motivation behind this thesis, it is first necessary to understand the directions that turbo code research has taken in the years since its first publication.

Two main component decoder strategies have been suggested for use with these codes, the Maximum *a posteriori* (MAP) algorithm and the Soft Output Viterbi Algorithm (SOVA). The originally proposed MAP algorithm and its derivatives (Max-Log-MAP and the later Log-MAP) receive by far the most attention due to the fact that their overall performance is an improvement over that of SOVA, generally showing around 0.5dB to 1dB reduction in the signal-to-noise ratio (SNR) required to attain low bit-error-rates (BER). Conversely, the complexity of SOVA is generally accepted to be around half that of the least complex of the MAP derivatives, Max-Log-MAP.

As well as concentrating on the MAP based algorithms, turbo code research generally considers long datawords (those with 1000 bits or more). There are two reasons behind this, one a bi-product of the other. First, the iterative nature of turbo codes does not immediately lend itself to applications, like voice calls, where short frames are used as, along with excellent error correction, it has inherently high latency. As this delay limits the uses of turbo codes to those applications where latency is less important and as longer frames give increased performance, research into turbo codes with longer frames would indeed be the direction to take, while waiting for Moore's law to reach a point at which the hardware is fast enough to consider other applications.

Turbo decoders require good knowledge of the transmission channel to perform as well as they do. Two pieces of information are needed, the SNR and the channel fading amplitude. The literature shows that the effects of bad SNR estimation is small for the MAP based decoders although no investigations have been made with regards to the effects on the SOVA decoder. The decoders do require an accurate channel fading value however, and it is here that problems can arise. Dispersion channels can produce large amounts of Inter-Symbol Interference (ISI), which obviously affects the quality of the channel fading amplitude estimates. To overcome this obstacle, the iterative theory proven with turbo codes has been applied to a combined equaliser/decoder system that works in much the same way as turbo codes but with one component decoder replaced with an equaliser. Information

is passed between the two devices in much the same way, with added complexity due to the repeated channel interleaving and pilot symbol re-insertion.

In the eleven years since their inception, turbo codes have been applied to only a very select number of applications. Due in part to the complexity of the decoders and the delay it produces. Modern, third generation, mobile communications systems now include turbo codes for data transmissions but still utilise convolutional codes for shorter frame transmissions such as those used for voice communications. With the increasing demands being made on mobile communications systems bandwidth is at a premium. It would therefore seem sensible to further investigate codes that require a minimum bandwidth and it is this that is the underlying motivation of the thesis.

When considering short frame applications, overall performance is not necessarily the most important factor. In these situations, latency is as much, if not more, of a factor, with BERs in the region 5×10^{-3} to 4×10^{-2} deemed to be acceptable [KOO97].

1.2 Aims of Thesis

This thesis examines the conventionally held beliefs concerning turbo codes and to determine whether they stand for short frame applications. It will also examine small, but significant areas that have not yet been investigated for SOVA decoders. It will investigate the effects of dispersion channels on turbo codes with short frames. To this end, the aims of this thesis are as follows.

To investigate the effects of Additive White Gaussian Noise (AWGN) and Rayleigh fading channels for short frame turbo codes in terms of bit-error-rate and frame-error-rate.

To compare the most commonly used MAP derivative decoder, Log-MAP, and SOVA for overall performance in short frame transmission systems.

To compare the same decoders at bit-error-rates suitable for voice quality transmission in terms of decoder complexity, as an indication of decoder delay, as opposed to on an equal iteration basis.

To investigate the effects of interleaver design and puncturing on turbo codes with frames of this size

To investigate the effect of errors in signal-to-noise ratio estimation at the SOVA decoder and the effect of errors in fading amplitude estimation at the Log-MAP and SOVA decoders for these codes.

To investigate the effects of static and European Telecommunications Standards Institute (ETSI) proposed dynamic indoor, pedestrian and vehicular channels on short frame turbo codes and investigate the use of adaptive equalisation techniques to improve the performance of these codes while maintaining low complexity.

1.3 Objectives

The objectives of this thesis are:

To choose, design and implement four turbo encoder schemes with different numbers of states and error correction capabilities for use within the scheme;

To design and implement interleavers suitable for these codes with frame sizes of 100 bits and 512 bits;

To investigate the complexity of the two simulated decoding algorithms;

To compare the two decoding algorithms for equal complexity;

To determine the results of inaccurate channel amplitude estimation at the turbo decoder;

To determine the effect of signal-to-noise ratio inaccuracies at the SOVA decoder;

To combine these techniques with SOVA and Log-MAP turbo decoders to produce a low complexity equaliser/decoder scheme;

To obtain results using this scheme over time invariant and time variant ISI channels.

1.4 Statement of Originality

The areas of this thesis that exhibit original and novel work begin with the investigation of these codes with short frames. Although a minimal amount of work has been published, no research exists that provides such a comprehensive investigation in both the number of codes researched and the variety of effects investigated, looking at the effects of different frame sizes, different channels, comparison of code rate reduction techniques, decoders, channel estimation errors, effects on frame error and bit-error-rates.

The 16-state component code used here has not been found in the literature and is therefore assumed to be novel, as is the 32-state component code with respect to short frame turbo codes. The 16-state component code is also optimal for component codes with this constraint length. Although a small amount of research has been conducted with respect to the comparison of turbo decoders for short frame applications, the literature does not include codes with component encoders of such varied or large constraint lengths. Nor has any research been published with as comprehensive a comparison of punctured and non-punctured codes in this area of turbo codes.

The investigation and comparisons of the number of operations per frame of turbo decoding algorithms for short frames is original, as are the investigations pertaining to BERs and FERs suitable to these applications along with the discoveries made.

The effect of errors in signal-to-noise ratio estimation at the turbo decoder have previously been made for the MAP based schemes, but the investigation into these effects on the SOVA scheme have never previously been published. The effects of fading amplitude estimation errors have also not been previously investigated for Log-MAP turbo codes with short frames or SOVA decoders at all.

The research into the effects of ISI on turbo codes is original. No investigation has previously been made into the effects of the ETSI mobile radio channels, much less with the modified and improved Power Spectral Density. Finally, the combination of adaptive equalisation techniques with turbo codes to counteract the effects of this interference is novel for both the Least Mean Square (LMS) and Recursive Least Square (RLS) algorithms.

This thesis begins with an overview of turbo codes. Chapter 2 looks at their history, development and the current state of the art. The chapter reviews the original, groundbreaking publications and those that have furthered the understanding of the subject before looking at some of the directions that turbo code research has taken in recent years.

Chapter 3 looks at the manipulation of analogue signals for transmission over mobile radio communications channels. The chapter then explores the various forms of interference that may occur when transmitting over such a medium and how this is represented and simulated for research.

Chapter 4 gives the theory of turbo codes, beginning by explaining the role and importance of each component in the encoder and decoder systems before providing full derivations of the main decoding algorithms.

Chapters 5 and 6 provide the originality in this research. Chapter 5 begins by examining the design decisions that must be considered when implementing turbo codes, followed by full worked examples of the decoder process for the two decoding algorithms simulated. Results are then presented for turbo codes with 4, 8, 16 and 32 states over AWGN and Rayleigh fading channels. An investigation of the decoder complexities is then made, with further results presented comparing each strategy on a complexity basis. Finally, investigations are also made into the effect of errors in SNR estimation at a turbo decoder using SOVA decoding.

Chapter 6 gives the theory of adaptive equalisation techniques and explains how these might be combined with the turbo decoder to improve performance in channels where ISI occurs. The chapter also gives justification for the use of these devices, rather than those used in other research.

The chapter then details the design of the systems combining turbo codes with equalisation techniques. Novel results are obtained for the two codes over static dispersion channels and the two common decoding algorithms. Results are also obtained for short frame turbo codes over dynamic mobile communications channels. The dynamic channels are based on the ETSI defined channels for the office, pedestrian and vehicular environments. These results are made more realistic through the inclusion of a more accurate Doppler spectrum[1]. The information gained from experimentation with the static dispersion channel, with regards to combining the turbo decoder with equalization techniques are also applied to these channels.

Finally, chapter 7 draws together the conclusions made throughout the thesis on the results obtained and makes some suggestions for further research.

At this point in time, two IEE/IEEE referred conference papers have been published ([DEN00],

[1] The investigation and creation of the ETSI channel software was undertaken in partnership with Cameron Shaw.

Chapter

2

Literature Survey

2.1 Introduction

This chapter reviews the history of turbo codes, showing the development of the tools required to create them and the advances made in the application and appreciation of a code that is still in its infancy. The chapter begins with the original publications on the subject and those that have inspired and influenced the design of these codes, without which the concept would never have been realised.
The chapter then goes on to review the design developments that have taken place over the last 11 years, examining the modifications made to the original components to further improve the error control capabilities and reduce the complexity, and the developments in understanding the relevance of each component and how it interacts with the whole.

The discussion then turns to areas examined to date. This section looks at other aspects and areas of research within the realm of turbo codes, for instance, the different types of channel that the codes have been tested against and the qualities these codes portray for terrestrial applications.

In conclusion, this chapter examines the future of turbo codes, looking at what aspects need to be further researched and improved as well as the new opportunities offered by turbo codes and the new applications of the 'turbo' type iterative system.

2.2 Classical Turbo Codes

The turbo code model was first unveiled in 1993 in [BER93]. In this paper Berrou *et al* laid the foundations for what have now become known as the classical turbo encoder and decoder structures.

The encoder consists of two identical convolutional component encoders, in parallel concatenation and separated by an interleaver, and the decoder comprises two Soft-Input Soft-Output (SISO) decoders in serial concatenation, passing information between one another in an iterative process.

Using recursive systematic convolutional (RSC) encoders as components within the turbo encoder meant that as well as improving the weight properties of the codes, the amount of information transmitted could also be increased. The component decoder described was a modified version of the BCJR algorithm [BAH74]. The paper showed how the algorithm should be altered for RSC codes and how the log-likelihood ratio (LLR) of each symbol was constructed in part from the input data but also from extrinsic information created within the decoder. The algorithm was then further developed to accept this extrinsic information, allowing each component decoder to use information created by the previous decoder, thus improving the likelihood of correctly decoded data. Performance was evaluated for long codeword frames without puncturing, with a pseudo-random interleaver for a high number of iterations. The authors recognised the complexity of the decoding algorithm and understood the need for a simpler algorithm, more suited to implementation.

Although turbo codes were first made public in 1993, the original design of the decoding algorithm used goes back three decades. In 1972, Bahl, Cocke, Jelinek and Raviv published their work [BAH72]. Coincidentally, McAdam, Welch and Weber published [McA72], detailing a similar algorithm but with respect to convolutional codes only, rather than all linear codes. Both papers, a fuller version of which is [BAH74], described a new decoding method that became known as both the BCJR and the Maximum *a Posteriori* (MAP) algorithm. Their maximum likelihood algorithm minimised the probability of bit or symbol error, unlike the Viterbi algorithm [VIT67], [FOR73], which minimises the probability of word error without necessarily minimising bit or symbol error.

The concepts of recursive and systematic convolutional codes have been known for many years. Berrou [BER01], [BER03], attributes the first discussion of recursive convolutional codes to Forney [FOR70] in his comparison with their non-recursive counterparts, although Viterbi [VIT71] later omitted to mention recursive codes in his comparison between systematic and non-systematic convolutional codes. Recursive systematic forms of convolutional encoders are based on pseudo-random scrambling techniques and it was random codes that were used by Shannon [SHA48] to derive the theoretical coding limit.

The components of the turbo coding technique have therefore been known and understood for many years, so it is more the combination of these that was innovative. However, understanding the components separately and understanding the relationship between them are two different things and it is this that the next sections of this chapter examine.

2.3 Investigating and Improving the Components of Turbo Codes

With the advent of a new coding scheme comes the inevitable investigation into what constitutes a good version of that code. The fact that the components of the turbo code were already well understood did not mean that this was any less important.

Although the results of [BER93] were exceptional, the design had been intuitive to an extent [BER03] and, as with many novel ideas, further investigation was necessary to fully understand and develop the codes. If these codes were to fulfil their potential, less complex versions of the decoder would also have to be developed.

This section looks at the publications that advanced the concepts, explored the interactions of the components and introduced less complex algorithms.

2.3.1 Developing the Turbo Decoder

From the beginning, it was known that the greatest hindrance to the practical use of turbo codes lay in the complexity of the decoding algorithm. Although far simpler than other codes of similar performance, the structure still required a high number of operations. This section of the chapter looks at the development of the decoding methods and the trade off between complexity and performance.

The Soft-Output Viterbi Algorithm (SOVA) was first described in [HAG89] and is described in detail in chapter 4.5.2. Originally suggested for the decoding of serial concatenated codes, the algorithm was a development of the famous Viterbi algorithm and returned the *a posteriori* probability for each bit in the sequence. With the advent of turbo codes, it was immediately obvious that this algorithm could be applied to the turbo system. The algorithm was first presented for use in turbo codes in [BER93a] and is explained in greater detail in [HAG96]. The advantage in this decoder design lies in its low complexity when compared with the modified BCJR algorithm of [BER93], however, this is offset by the reduction in performance, in part due to the fact that the algorithm still works to reduce the probability of codeword error, rather than symbol error.

There is also, when regarding the application of SOVA to turbo decoding, the question of the algorithms tendency to over estimate the reliability associated with each data bit. In [PAP96], the authors showed that the algorithm actually suffers from two distortions. They proved, using a Gaussian approximation of the received data, that the first distortion is inherent in the SOVA system and is a multiplicative factor, dependent on the current bit-error-rate, requiring a normalising factor to approach the true log-likelihood ratio. The second is a correlation between the *a priori* information passed from the previous decoder and the intrinsic information in the received data. Since these two streams are regarded as uncorrelated by the SOVA decoder, a correcting term is necessary. The authors showed that it is the fact that the SOVA is sub-optimal and only considers two paths when determining the soft

output which causes this correlation. With SOVA, there is a possibility that the competitor path used is not the 'true' competitor path since this may have been eliminated and as such, the competitor path used to deduce the reliability value will have a lower path metric than the 'true' competitor, implying a higher soft output and therefore a higher reliability estimate. Using the MAP algorithm as a benchmark, the authors showed that the gain of the MAP algorithm over SOVA was reduced when using their normalisation procedure and that this could be further reduced by including correlation compensation.

The authors of [LIN97] compared the SOVA updating procedures of [HAG95] and [BAT87], as the normalisation factor of [PAP96] was quite complex and required knowledge of the signal-to-noise ratio. The procedure according to [BAT87] is to compare the survivor bit with the competitor bit at each time step. If the bits do not match, the bit reliability should be updated to the smaller of two values, the path reliability and the survivor metric. On the other hand, if the two bits match the update should be the value of either the survivor metric or the sum of the path reliability and the competitor metric, whichever is smaller. Hagenauer and Robertson's [HAG95] procedure only updates when the bits under comparison do not match and for this scenario the update is the same as that in [BAT87]. The authors of [LIN97] found that Battail's [BAT87] procedure was better but that Hagenauer's solution was easier to implement.

In 1994 Erfanian *et al* published [ERF94], proposing the max-log-MAP algorithm. A much-simplified version of the MAP algorithm, this version applies logarithms to Berrou's BCJR algorithm and approximates the final log-likelihood using a maximum function. The result being that the multiplications of the MAP algorithm become additions in the log domain, greatly reducing the algorithms complexity, with the maximum function further increasing the simplicity. Although sub-optimal, the results obtained using this algorithm were an improvement on those of the SOVA and a simplification of the operations of the MAP algorithm of [BER93].

In [ROB95], Like Berrou *et al*, the authors understood that MAP was too complex due to "numerical representation of probabilities, non-linear functions and the high number of additions and multiplications". They also saw that Max-Log-MAP and SOVA were sub-optimal, especially at low signal-to-noise ratios where turbo codes are of greatest benefit. The paper proved that Max-Log-MAP and SOVA made the same hard decisions but that they used reliability information in different ways, SOVA comparing the survivor path with only one competitor, itself a survivor of the Viterbi algorithm, while Max-Log-MAP looked at two paths per transition (MAP takes all paths into account, hence it is optimal), the best 'one' path and the best 'zero' path, the difference of their log-likelihoods being the output. A comprehensive comparison of complexity was also given for the first time, but the most important aspect of the paper was the introduction of the Log-MAP algorithm which, rather than using the max approximation of the Max-Log-MAP algorithm, used the Jacobian algorithm, which contains a correction function that improves upon the max operation, to find the correct log-likelihood for each symbol. The complexity comparison showed that the order of complexity (most to least) was MAP, Log-MAP, Max-Log-MAP and finally SOVA. Simulation results showed that log-MAP and MAP performed almost equally in terms of bit error rate performance and, consequently, Log-MAP was an improvement upon Max-Log-MAP.

2.3.2 Investigating the Turbo Encoder

Accompanying the developments of the decoding system was the exploration into why these codes perform well. This section analyses the structure of turbo codes to show how it was developed, how the understanding of the system matured and how this led to improved turbo code design.

The first development of the turbo encoder was by [JOE94] and later, [ROB94]. These authors highlighted the necessity for trellis termination, a subject omitted from the original publication. It was understood that, unlike non-systematic convolutional codes which can be forced to the all-zero state with only zeros, the RSC codes required different termination bits to be appended dependent on the final state of the encoder. Realising the optimal solution would be that both component trellises were terminated, but also that the presence of an interleaver in the turbo encoder meant that any tail bits appended to the data stream with regards to terminating the first encoder trellis at the all-zero state were unlikely to terminate the second trellis, [ROB94] showed a practical, though sub-optimal, solution by terminating the first component encoder to ensure that the trellis ended at the same state that it began. The authors showed experimentally that terminating only the first encoder output and leaving the second trellis 'open' had only a small effect on turbo codes with sufficiently large frame sizes. [ROB94] was also amongst the first to show that the performance of turbo codes was affected by the construction of the interleaver, explaining how weight-2 input frames (the minimum weight of data that can cause an RSC encoder to diverge from the all-zero state and converge at some later point) that produced a low weight output could be permuted by a poorly chosen interleaver into another input that again produced a low weight codeword, thus counteracting one of the main objectives of the turbo encoder. To avoid this, the authors described a rather intensive approach to improving the interleaver, the system being based on the observation of all information sequences that cause low codewords, one after another and rearranging the interleaver to suit. The system was longwinded but did improve the flattening effect, or error floor, caused by less well-designed interleavers.

With regard to hardware implementation of the turbo encoder, [DIV95] gives a simple solution to the termination problem (for a single component encoder). The authors add a switch between the input and the feedback loop. When receiving data, the encoder resembles the normal RSC encoder and for termination the feedback loop of the component encoder becomes the input to the encoder. Simple and effective, this system requires m bits, where m is memory length, to return to the all-zero state.

[DIV96], [BEN96] and [BEN96a] defined the effective free distance of a turbo code. Not to be confused with the minimum distance of a convolutional code, but having a similar effect for turbo codes, they showed it to be a function of the minimum parity weight caused by a weight-2 input to a turbo encoder and that to get the most out of a component code, especially at high signal-to-noise ratios, this value must be maximised. [BEN96a] also defined the maximum effective free distance for a RSC encoder with a single input stream and a particular encoder memory and produced a table of component convolutional codes that exhibited the best effective free distance and free distance for memory sizes ranging from 2 to 5. [DIV96] extended these tables to include multiple inputs.

[PER96] explained the reasons behind the performance of turbo codes by showing the ability of the codes to cause “spectral thinning” [PER96], an expansion of the ideas in [ROB94]. This thinning of the distance spectrum, brought about by the presence of the interleaver, means that datawords that produce low weight outputs are likely to be permuted such that the new dataword produces a high weight output. Therefore the turbo codewords would consist mostly of average-weight outputs with a small number of low-weight outputs. The authors showed that “spectral thinning is enhanced by increasing interleaver lengths” [PER96] and that this would also lower the error floor for a fixed free distance. Conversely, increasing the free distance of the component codes could also lower the error floor.

[HO98], [BEN98] and [HO98a] also investigated the effects of the distance properties of convolutional codes, producing extended results on previous papers for varying rates and memory sizes. Rather than simply examining the free distance and effective free distance, the authors of these publications widened the search for good component codes by investigating the qualities of codes for higher weight inputs. They also showed that the number of codewords with these distances was also an important factor when looking for the optimum component code and that the lower the number of possible codewords with these weights, the better the code performed.

Yuan *et al* showed, [YUA99], that the distance spectrum also plays an important role in designing the turbo encoder, especially at low signal-to-noise ratios. They show that choosing a component code with the smallest error coefficients (Where an error coefficient is the average number of bit errors caused by code words of a particular weight and determines the contribution of those code words to the bit error probability) for a given interleaver size and for low to medium Hamming distances can outperform other codes in low to medium signal-to-noise ratios, even when the effective free distance is not optimal.

In the research to understand the qualities of the turbo code structure it has also been necessary to investigate the error bounds of these codes. In order to determine the upper performance bounds for turbo codes [BEN96] treated the code as a systematic block code. The problem, however, was the inclusion of the interleaver. It is possible to determine the weight of the first parity sequence if the conditional weight enumerating function (WEF) is known, but the second parity sequence is not only dependent on the input weight but also on the sequence of the data after interleaving. [BEN96] proposed the use of a uniform interleaver, a probabilistic device based on an analysis of all possible interleavers, and showed that the method could be used to assess the bit error probabilities of turbo codes independent of the interleaver. As a result of this, the effect of the interleaver on the turbo encoder could be assessed, and therefore its gain. The authors showed in their analysis and proved by experimentation that the bit error probability reduced as interleaver size increased.

2.3.3 Interleaver Design

Early publications [ROB94], [JUN94], had shown that poorly chosen interleavers could reduce turbo code performance, while [BEN96] had shown the gains possible through correct choice of

interleaver. In light of these discoveries, much research was made into the method for finding, and the design of, the 'optimal' interleaver, some more involved [HOS00], [DAN99], than others.

Initial publications were basically trial and error based interleavers [BER93b], [JUN94], helping to show the improvements that were possible. Since then, many researchers have published papers on this aspect of turbo codes, some defining particular interleaver designs, others explaining systems that can be used to obtain good interleavers.

[DIV95a] defined "S-random" interleavers, where S = $\sqrt{N/2}$ and N is the size of the interleaver. The method selects a random integer value within the interleaver size and compares it with S previously selected integers. If the chosen integer is equal to, or within $\pm$ S positions, of one of the previous integers selected, then it is discarded. The process is repeated until all integers have been chosen.

[YUA99] defined the "Code Matched Interleaver" or CMI, where the authors compute the weight spectrum of the lower weight codewords then use performance analysis to determine the inputs that make large contributions to the error probability at high signal-to-noise ratios. The interleaver is then designed such that these patterns are not present after interleaving. Modifying the S-random interleaver of [DIV95a] after the comparison with previous integer choices, the system checks whether the interleaver produces an output that is undesirable. That being the case, the integer is again rejected and a new one selected. If no integer can satisfy both the comparison with previous choices and the output word control, the value of S is reduced by one and a new interleaver is searched for. Used in conjunction with good component codes, this system significantly lowered the error floor of previous designs.

[BYU99] also further developed the S-random interleaver. The authors pointed out that, for frame sizes larger than around 1000 bits, it is almost impossible to use an S value as described by [DIV95a]. Their design, called the swap interleaver, begins with a block interleaver of equal depth and span. Two random positions within the interleaver are chosen and swapped. These positions are then checked against the given S value and if this is not satisfied they are returned to their previous positions. After a sufficient number of iterations (the authors propose 100 times the frame length) the design is complete. The authors found that the search time was vastly reduced, especially where large values of S were desired and that the performance was in excess of that of the standard S-random interleaver.

[HO98b] looked at interleavers for punctured turbo codes, noting that puncturing often degrades the performance of turbo codes. The authors show that this is due to uneven parity bit protection. For example, using the common turbo code puncturing method (delete all even bits from the first parity bit stream and all odd bits from the second parity bit stream) may mean that one particular data bit has two parity bits, whereas another has none. This is easier to comprehend if one assumes that the position of the data bit prior to interleaving was in an odd position and after interleaving was in an even position, in which case, that particular information bit has a corresponding parity bit in both received codewords. To rectify this, the authors introduce the mod-*k* interleaver an extension of the odd-even interleaver (mod-2) described in [BAR94]. The odd-even interleaver permutes odd data bits

to odd positions and even data bits to even positions, thus ensuring that all data bits will be transmitted with one parity bit after puncturing. The authors of [HO98b] also considered the fact that most interleavers require a corresponding de-interleaver, which, in implementation, doubles the storage. To combat this, they propose the use of symmetric interleavers, one piece of hardware or look-up table that both interleaves and de-interleaves. Through experimentation it was proved that a mod-*k* interleaver could bring about an improvement in performance and that combining the two theories (symmetric and mod-*k*) increased performance even more. Comparing interleaver designs, the gain of the S-symmetric-mod-2 interleaver was shown to improve turbo code performance when compared with its S-random counterpart.

While there are many other publications available, detailing new and innovative interleaver methods for use with turbo codes in recent years, it is perhaps more relevant to review one other aspect of these components which is particularly useful for obtaining the very best bit-error-rates possible.

As discussed earlier, [ROB94] highlighted the fact that to get the most from the turbo encoder it must be designed such that both component encoders return to the all-zero state. With this in mind, two publications offer some insight into accomplishing this through cunning arrangement of the interleaver. [BLA95] expanded upon ideas put forth by [BAR95] and their helical interleaver design, to show how "the state variables at a specified end time depend only on the sum of message bits found in disjoint partitions of the message stream" and that, by interleaving within these partitions, both component encoders can be made to terminate at the same state. The authors describe the conditions that must be met to achieve this and show how this can be achieved in the helical interleaver of [BAR95].

[BRE99] developed this further to show a class of interleavers with this property. The authors show that the state at which the first component encoder terminates will be the state at which the second component encoder terminates, as long as an interleaver from this class is used. [BRE99] shows that the 'simile' interleavers of [BAR95] are a particular set from this class, with period equal to the component encoder memory plus one, and that helical interleavers are a subset of these. As another example, the authors give the 'rectangular' interleaver and show that this also has the even parity protection as defined in [HO98b]. They show by experimentation that this type of interleaver can outperform a helical interleaver of similar dimensions if designed correctly. The authors do concede, however, that the interleavers are not optimised with respect to the distance spectrum.

2.4 Short Frame Turbo Codes

As can be seen from the literature reviewed above, most turbo code research deals with frames with lengths of more than 1000 bits. Research into smaller frames, with applications for voice and other delay-constrained scenarios, tends to be avoided due to the inherent long decoding time that turbo codes carry in the decoding process. Not only are the component decoders more complex than Viterbi type decoders, the iterative nature of the turbo decoder and repeated interleaving and de-interleaving

create quite a handicap in terms of delay. To clarify, the number of operations performed by each component decoder within the turbo decoder are greater, per received bit, than for the Viterbi decoder. This coupled with the fact that each of the component decoders performs these operations on each bit once per iteration and that multiple interleave/deinterleave operations are performed per iteration, means that a given processor would require a much longer time to apply turbo decoding to a given frame than Viterbi, allowing equal complexity for each operation.

For the most part, therefore, research into short frames is relatively scarce and researchers look to increasing the error control capabilities of the code with little regard for latency, which "…is a weak point for turbo codes…" and "…is in fact the reason why a simple convolutional code was preferred in 3G voice transmission" [BER03]. This section details publications that have investigated turbo codes with short frames specifically and some that contain mention of them.

[JUN94a] was one of the first investigations into small frame turbo codes, experimenting with interleaver design by iteratively searching for interleavers that produce a low number of low-weight output sequences. This paper was also the first to publish frame-error-rate curves and showed that bit-error-rates suitable for voice transmission could be obtained at a signal-to-noise ratio of around 1.3dB for short frames. However, this was with a high number of iterations and only the MAP algorithm was considered.

[JUN96] is another of the earlier publications specifically investigating short frame turbo codes and giving a thorough examination. The paper looks at their application for voice transmission in Code-Division Multiple Access (CDMA) schemes and examining the results of codes over AWGN and fully interleaved flat Rayleigh channels. Expanding on earlier research, [JUN94], which showed that a block interleaver could produce satisfactory results for frame sizes less than 200 bits long, the author investigated Berrou *et al* MAP decoder, Robertson *et al* Log-MAP decoder, Erfanian *et al* max-Log-MAP decoder and the SOVA derived from Hagenauer *et al* work. Encoder constraint lengths ranging from 3 to 5, with the second component code only terminated were reviewed. Frame lengths of 192 bits with ideal Channel State Information (CSI) were assumed and ten decoder iterations were used. Interesting to note is the fact that in general, the Log-MAP algorithm equalled or outperformed the MAP algorithm, especially at higher SNRs, with max-Log-MAP and SOVA closely following.

[HAL98] produced some results for fully interleaved, slow Rayleigh fading channels pertinent to this section. The author's results were for simulations with side information, and without, of rate 1/3 turbo codes based on the $[21;37]_8$ component codes that were used in [BER93]. Log-MAP decoding was used and 15 iterations were performed.

[BER96] also looks at smaller frame sizes, showing how to avoid the termination of the component codewords and producing results over AWGN for a rate 1/2 turbo code, composed of two $[37;25]_8$ RSC encoders.

[KOO97] made an investigation into turbo codes with short frames, comparing turbo codes with different frame sizes. Un-punctured, 16-state codes were used with random interleavers and simulations were made for ten decoder iterations, over AWGN and Rayleigh channels assuming perfect channel state information.

2.5 Practical Issues for Turbo Codes

When considering turbo codes for practical communications systems, two important criteria must be considered. These codes provide excellent error control properties, but are dependent on knowledge of signal-to-noise ratios (SNR) and fading amplitude values. At least, this is the case theoretically. [VAL01a] and [WOR00], amongst others, showed that the MAP based decoders are not as susceptible to errors in SNR estimation as would be expected. The authors show that the Max-Log-MAP algorithm can be defined without any consideration of the relative signal energy and that the Log-MAP algorithm suffers only slightly from inaccuracies in the stated SNR.

The other criteria, that of the fading amplitude of the channel is a separate issue. Under Added White Gaussian Noise (AWGN) conditions, the fading amplitude is unity and is therefore of no real concern. However, under real transmission conditions, the fading characteristics of a channel play an important part in the attenuation undergone by the transmitted signal. There is an abundance of research with regards to the effects of fading channels when channel state information is assumed to be perfect, [HAL98], [VAT02], and [TAN99].

More recent research examines various methods for the detection of the channel state information (CSI) in combination with the turbo decoder. These methods range from the simpler techniques, such as the use of finite impulse response (FIR) and least mean squared (LMS) filtering techniques to estimate channel amplitude prior to turbo decoding, [VAL98], [KAZ99] to more involved techniques like [ANG01] and [ANG02], which use a variable step size LMS algorithm in conjunction with the turbo decoder to iteratively detect and improve channel estimation, reconstructing the transmitted signal after each turbo iteration to re-evaluate the channel before proceeding to the next iteration.

The advent of the turbo code brought about a wide spread re-think of many systems in the communications community and the turbo principal has been applied to other systems such as joint source-channel decoding ([HAG01], [HAG03]), joint channel estimation or equalisation and decoding ([BAU98], [LIU04], [TUC02]) and detection and decoding on multi-user and spread-spectrum communications systems ([VAL01], [DAI02]).

One system that neatly bypasses some of the problems encountered in applying turbo codes to real communications systems is turbo equalisation [BAU98], [LIU04]. Turbo equalisation utilizes only one convolutional encoder with an interleaved output, and regards a channel with inter-symbol interference (ISI) as the second (or ‘inner’) decoder of a serially concatenated coding scheme. At the receiver, a SISO equaliser is used together with a SISO decoder in an iterative feedback system, following the turbo principle. In this way it can be seen that the systems of [ANG01] and [ANG02] are in effect taking the turbo equalisation scheme one step further with two feedback loops, one within the turbo decoder and one between the turbo decoder and channel estimator.

2.6 Applications of Turbo Codes

After eleven years of investigation into the properties and design of turbo codes, their incorporation into mobile communications schemes is a reality. [BER03] details six separate, current, applications of turbo codes for low bit-error-rate situations such as digital television, video on demand and video conferencing (Digital Video Broadcasting-Return Channel Satellite (DVB-RCS), Digital Video Broadcasting-Return Channel Terrestrial DVB-RCT, INMARSAT and EUTELSAT), and deep space communications (The Consultative Committee for Space Data Systems (CCSDS) standards). Alongside these is the application of turbo codes in mobile communications, specifically the Universal Mobile Telecommunications System (UMTS) CDMA2000 applications for low BERs [VAL01b]. There are no applications for voice or shorter frame codes as [BER03] points out, mainly due to the decoding delay of these codes.

2.7 Conclusion

This chapter has looked at the development of turbo codes since their inception. Beginning with the ideas behind the system and outlining the development into one of the most powerful codes available today. Discussion began with the classical design and continued through the improvement of each of the basic components, together with the analysis tools required for proper investigation.

The theory of turbo codes is now widely understood. From intuitive beginnings researchers have explained the reasons for the remarkable error control properties and found ways to estimate the peak performance of these codes. In reaching for this performance, the design and optimisation of the components of the encoder and decoder structures have been well examined. It is now possible to determine the best component encoder using similar methods to those used for convolutional codes. Research has also produced some excellent interleaving techniques, neatly counteracting the effects of puncturing and component code termination. Decoding methods have been proposed and elaborated upon and researchers now understand the processes and pitfalls of each algorithm.

Certain problem areas still remain however. Although the error control capabilities of these codes are well known, so are the delay problems. It has been stated that these codes require a further increase in Moore’s law before they become viable options in many applications. Indeed, those applications that have become reality are less time constrained and sacrifice decoder delay in favour of performance.

Originally, comparisons of decoding algorithms looked at performance differences of around 0.5dB with little regard to the relative complexities of these decoders. Research, in the main, looked to long frame lengths and high numbers of iterations to attain better and better performance. As a consequence of this, it appears that turbo codes have been disregarded for use in short framelength applications with little investigation. The fact that the highest performing turbo decoder is restrictively

complex should not mean that turbo codes are unsuitable for these situations as overall performance is not necessarily the issue. Although one code may not exhibit the absolute best performance for a particular SNR, it may still be chosen if it uses less bandwidth or incurs an acceptable delay. In the case of modern mobile voice communications, with increasing numbers of users, a code may well be chosen on the amount of bandwidth it uses as long as it reaches an acceptable BER and the delay is not discernible.

A small amount of research has been made into turbo code suitability for these applications, although particular areas of the codes have not been addressed. For instance, no information exists on the effects of puncturing in these applications. Nor have they been compared with anything other than block interleavers. The only available research with respect to complexity is to compare the MAP algorithm with convolutional codes. Besides the fact that this algorithm does not lend itself to implementation, it is also the most complex decoder algorithm available and incurs a much longer delay than other algorithms.

Another area that has received little or no attention is the effect of ISI on these codes. It has been shown that the MAP algorithm and its derivatives are not particularly susceptible to errors in SNR estimation, although the loss in performance is around the same as that when comparing SOVA and MAP with ideal channel information, 0.5 to 1dB. No equivalent experiments have been conducted for SOVA. It is widely reported that accurate channel amplitude information plays a far greater part in the performance of turbo codes and this is where problems arise when considering channels with ISI. The received symbols are not only a product of the fading amplitude and the transmitted symbol, but are also affected by other symbols transmitted before and after, themselves affected by fading too. To improve performance over channels exhibiting these properties, the research community has applied the main principle of turbo codes, the iterative feedback at the receiver, to the combination of SISO equalisers and decoders. This method has received much interest in recent years and definitely has promise for combating severe ISI. The downside, however, is that this system has a greater delay than turbo codes. The equaliser can be looked at as a second component decoder, but to accomplish the feedback between the two components, interleaving and pilot symbol re-insertion must also take place. The delay involved is not a major problem for those applications that already use turbo codes, but as with the original turbo code publications, it does not solve the problem for short frame applications.
[DEN04]).

Chapter

3

Wireless Channel Theory

3.1 Introduction

Before any investigations can take place into the effectiveness of turbo codes, an understanding of the conditions they must endure is first necessary.

The channel is the medium through which a signal is broadcast. No matter what method is used to pass information from one position to another, the channel will distort the transmitted message. The neutralisation of this distortion determines the effectiveness of a transmission system.

The most common type of distortion occurring in a transmission channel is additive noise, which limits the rate of message transfer and the range of a successful transmission. The performance of a receiver is governed by the signal-to-noise ratio (SNR) and it is therefore imperative that this be determined at the receiver.

In mobile communications, another effect commonly encountered is fading. This is caused by multi-path propagation due the surrounding environment. Figure 3.1 shows a typical example of the conditions encountered in mobile communications. As in the diagram, often there is no line-of-sight (LOS) between the transmitter and receiver, implying that the information is received by way of reflections off objects (stationary or mobile), diffraction in the atmosphere and scattering. These signals arrive at the receiver with varying delays, amplitudes and phases and can therefore cause all kinds of problems.

A communications system does not consist solely of the error control code. Turbo codes produce digital signals, which are discrete. Transmitted signals are analogue and continuous. It is therefore necessary to convert the coded digital information into an analogue signal and the device that performs this task is the modulator.

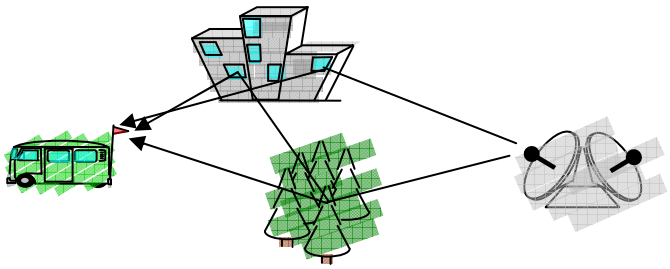


Figure 3.1: Multi-path propagation

This chapter describes the modulation method used in this thesis. Following an explanation of this device, derivations and definitions of the mathematical models of the channels encountered in this research are given. The effects that mobile radio channels have on transmitted data are described as well as the random processes that are used to simulate them. Explanations of the alterations possible within the processes and how these change the channel properties are also discussed.

3.2 Modulation

As mentioned in the introduction, the modulator is the interface between the digital error control coding and the channel. In general terms, the device accomplishes this task by mapping a group of bits, where the number of bits in the group can be said to be $n = log_2 M$, from the code word onto one of $M = 2^n$ pre-determined waveforms with finite energy, ready for transmission.

The set of waves associated with the modulator can differ in amplitude, frequency or phase and it is the latter, phase modulation, which is described here.

3.2.1 Binary Phase-Shift Keying

For all types of M-ary modulation, there are $M=2^n$ possible waveforms, or signals. Symbol duration is denoted as $T_s=nT_b$, where T_b is the bit duration. The bandwidth necessary for transmission of the signal is inversely proportional to the signal time, therefore:

$$B = \frac{1}{T_s} = \frac{1}{nT_b} \tag{3.1}$$

Where B is the channel bandwidth. From (3.1) it is obvious that the higher the number of bits associated with the transmitted signal, the lower the necessary bandwidth.

To understand the modulation mechanism, first consider the transmission system model of figure 3.2.

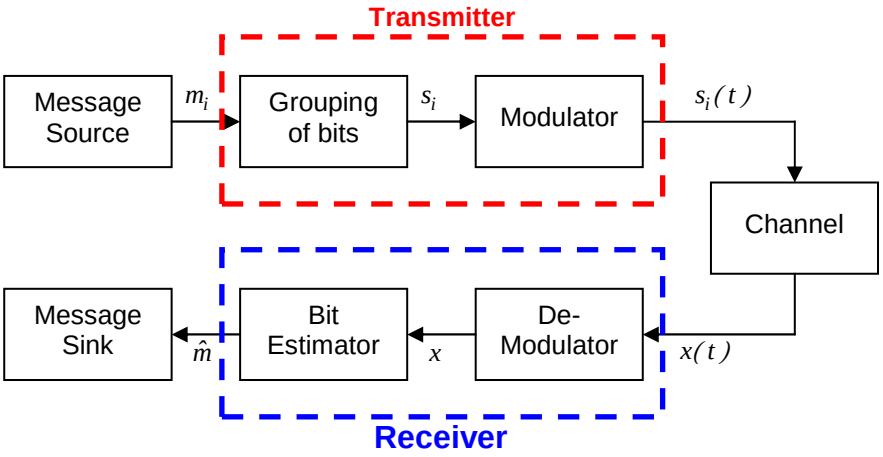


Figure 3.2: Transmission System Model

In the figure above, the message source emits one symbol every T_s seconds belonging to an alphabet of M possible symbols. Each possible symbol has a probability of occurring associated with it. Generally, all probabilities are equal, thus:

$$p(m_1) = p(m_2) = \ldots = p(m_i) = \frac{1}{M}, \text{ for all i} \tag{3.2}$$

In M-ary terms, the output of the message source is then grouped into vectors s_i with n real elements. Each vector corresponds to one of the M symbols. Note that $n \leq M$. The modulator then creates a distinct signal $s_i(t)$ with duration T_s. The signal $s_i(t)$ is therefore the representation of the message m_i. This signal has energy:

$$E_i = \int_0^T s_i^2(t)dt \text{ , } i = 1,2,\ldots,M \tag{3.3}$$

At the receiver, the modulator reverses the operations made in the transmitter.

In the case of Binary Phase-Shift Keying (BPSK), the value of M is obviously 2 and therefore n = 1. As the name suggests, the step change that distinguishes one signal from another occurs in the phase of the signal. The signals $s_1(t)$ and $s_2(t)$ correspond to binary 1 and 0 respectively and are represented thus:

$$s_1(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\pi f_c t) \tag{3.4}$$

$$s_2(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t + \pi) = -\sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t) \tag{3.5}$$

Where $0 \leq t \leq T_b$ and E_b is the transmitted signal energy per bit. The carrier frequency f_c is made equal to n_c/T_b where n_c is an integer, in order that an integer number of cycles of the carrier wave are used to represent each bit. The signals $s_1(t)$ and $s_2(t)$ are referred to as *antipodal* as they are phase shifted by 180˚. From equations (3.4) and (3.5), it can be seen that for BPSK, there is only one basis function for unit energy and that is:

$$\phi_1(t) = \sqrt{\frac{2}{T_b}}\cos(2\pi f_c t),\ 0 \leq t \leq T_b \tag{3.6}$$

Thus, the two signals can also be expressed as:

$$s_1(t) = \sqrt{E_b}\phi_1(t),\ 0 \leq t \leq T_b \tag{3.7}$$

$$s_2(t) = -\sqrt{E_b}\phi_1(t),\ 0 \leq t \leq T_b \tag{3.8}$$

A coherent BPSK signal is therefore characterised as having a one-dimensional signal space with a signal constellation made up of $M = 2$ message points with the following coordinates:

$$s_1 = \int_0^{T_b} s_1(t)\phi_1(t)dt = \sqrt{E_b} \tag{3.9}$$

$$s_2 = \int_0^{T_b} s_2(t)\phi_1(t)dt = -\sqrt{E_b} \tag{3.10}$$

Figure 3.3 shows a BPSK constellation with minimum average energy.

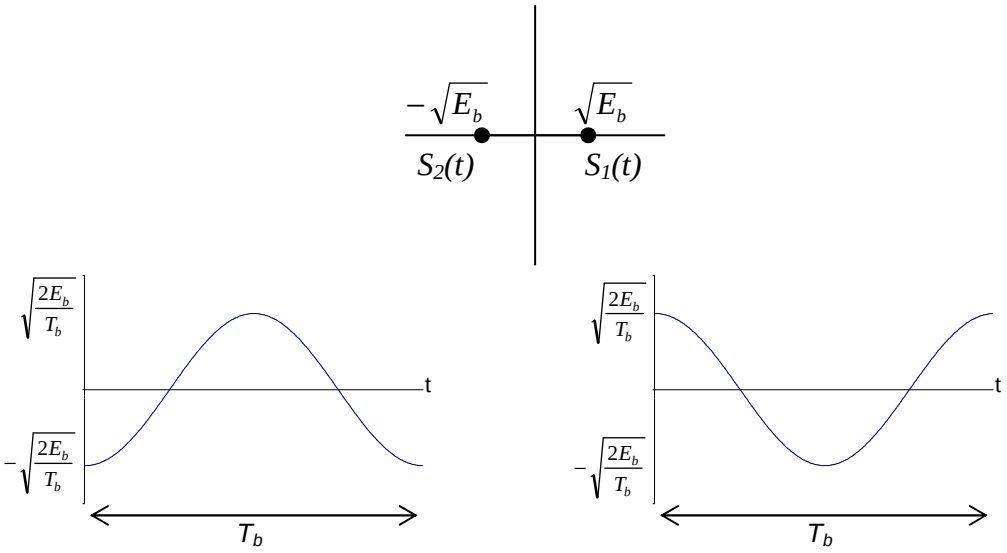


Figure 3.3: BPSK Constellation and Antipodal Signals with Number of Wave Cycles per Transmitted Bit (n_c) Equal to 1

3.3 Random Processes

Since all transmitted signals are subject to perturbation across the channel, the analysis of communications systems must take it into account. To simulate the effects of the distortions encountered by these signals, the transmission media must be modelled. This section describes two random processes that are commonly used in the study of channel effects on communications systems. Later sections will define the situations and models in which these processes are involved.

3.3.1 The Gaussian Process

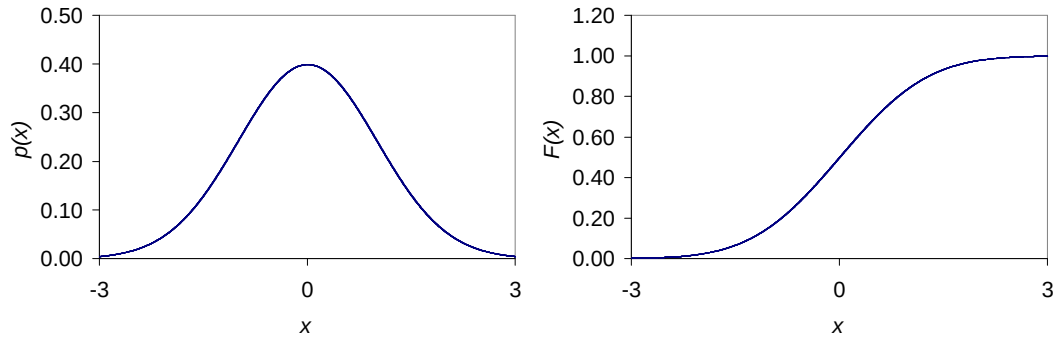
If a random variable *X* is a linear functional of *Y(t)*, defined as:

$$X = \int_0^T g(t)Y(t)dt \tag{3.11}$$

Where the weighting function *g(t)* causes the mean square of *X* to be finite and *X* has a Gaussian distribution for every *g(t)*, then *Y(t)* is a Gaussian process. The variable *X* has a Gaussian distribution if the probability density function is:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]} \tag{3.12}$$

Where μ is the mean and σ^2 is the variance. A Gaussian distribution is said to be normalised when it has a mean of zero and a variance of 1, giving a distribution such as that in figure 3.4a below.



(a) (b)

Figure 3.4: Normalised Gaussian PDF (a) and CDF (b)

The Cumulative Distribution Function (CDF) is defined as:

$$F(x) = \int_{-\infty}^{x} p(u)du$$

$$F(x) = \frac{1}{\sqrt{2\pi\sigma}} \int_{-\infty}^{x} e^{-(u-\mu)^2/2\sigma^2} du$$

$$F(x) = \frac{1}{2}\frac{2}{\sqrt{\pi}} \int_{-\infty}^{(x-\mu)/\sqrt{2}\sigma} e^{-t^2} dt$$

$$F(x) = \frac{1}{2} + \frac{1}{2} erf\left(\frac{x-\mu}{\sqrt{2}\sigma}\right) \tag{3.13}$$

Where *erf(x)* is the error function:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt \tag{3.14}$$

3.3.2 The Rayleigh Process

Formed by the combination of two Gaussian random variables, the Rayleigh process is defined thus:

Assume two statistically independent Gaussian random variables, X_1 and X_2, both with zero-mean and variance σ^2. A Rayleigh random variable is defined as:

$$R = \sqrt{X_1^2 + X_2^2} \tag{3.15}$$

The PDF is defined as:

$$p(r) = \frac{r}{\sigma^2} e^{-r^2/2\sigma^2}, \; r \geq 0 \tag{3.16}$$

The corresponding CDF is:

$$F(r) = \int_0^r \frac{u}{\sigma^2} e^{\frac{-u^2}{2\sigma^2}} du$$

$$F(r) = 1 - e^{\frac{-r^2}{2\sigma^2}}, \; r \geq 0 \tag{3.17}$$

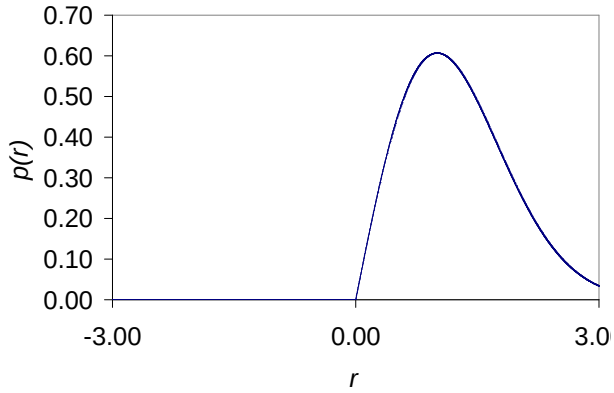


Figure 3.5: Normalised Rayleigh Probability Density Function

3.4 Gaussian Noise

The Additive White Gaussian Noise (AWGN) channel represents the effects of component noise and interference from other signals on the transmitted signal. It is applied in communications research to define the most basic mathematical model for a communications channel. This type of noise has a continuous and uniform frequency spectrum over its specified bandwidth and is easy to work with due to the fact that it is completely defined by the mean and variance. The fact that it is simple and that it is present in many communications channels means that it is commonly used for comparative analysis of communications schemes.

3.4.1 White Noise

'The adjective *white* is used in the sense that white light contains equal amounts of all frequencies within the visible band of electromagnetic radiation' [HAY01]. The power spectral density of white noise (not just Gaussian white noise), with a sample function *w(t)*, is:

$$S_W(f) = \frac{N_0}{2} \tag{3.18}$$

This is shown in figure 3.6a below. The autocorrelation function is the inverse Fourier transform of the power spectral density and is therefore defined as:

$$R_W(\tau) = \frac{N_0}{2}\delta(\tau) \tag{3.19}$$

Note that the autocorrelation function of white noise is zero when τ is not zero. This implies that two samples of white noise are correlated as long as they are separated in time. This information applies to all white noise and applying it to Gaussian noise means that any two samples of white Gaussian noise would be both uncorrelated and statistically independent.

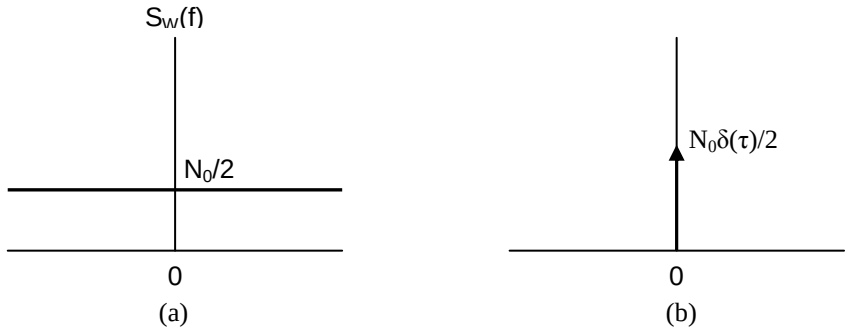


Figure 3.6: White Noise Characteristics, (a) Power Spectral Density and (b) Autocorrelation Function

As the name suggests, added White Gaussian Noise is summed with the transmitted signal after modulation as shown in figure 3.7.

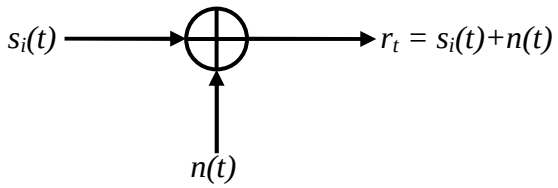


Figure 3.7: Combining AWGN with Transmitted Signal

The noise sample *n(t)* is a sample function of the added white Gaussian noise process with mean of zero and power spectral density = $N_0/2$.

3.5 Fading

As mentioned earlier, a radio communications channel is not only subject to the effects of noise, but also fluctuations caused by multi-path propagation. These effects come under the banner of fading, which itself can be split up into large-scale fading and small-scale fading.

Large scale fading is an average signal power attenuation or path loss caused by motion over large areas caused, for instance, by variations in terrain such as hills, built up areas and forests. Small scale fading denotes changes in amplitude and phase of signals as a result of small changes in the spatial separation of the receiver and transmitter. When these effects are due to temporal spreading they can be of the flat fading variety or the frequency selective type, whereas fast fading and slow fading would describe time variations in the channel.

There are three basic phenomena encountered by a radio signal. The first, reflection, occurs when large, smooth, surfaces redirect the signal. Diffraction, the second, is caused by large, dense,

objects blocking the line of sight path between the transmitter and the receiver. This has the effect that secondary waves to form behind the object. The final phenomenon, scattering, is a result of the transmitted signal rebounding off large, rough surfaces, spreading the signal energy out in many directions.

3.5.1 The Multi-Path Effect

To explain the effects of multi-path propagation, the example of multiple impulses, transmitted across a theoretical channel, is often used. Consider figure 3.8 below, giving a stylised representation of received signals resulting from transmitted impulses at different times. As can be seen, the multiple paths taken result in a train of pulses. This is the first characteristic of multi-path channels, the 'delay spread' introduced to the signal. The impulse transmission is then repeated. The time variations in the structure of the transmitted medium (moving vehicles, wind blowing trees etc) cause the paths taken to alter, thus altering the received pulse train. These alterations can vary the size, number and relative delays of each of the pulses.

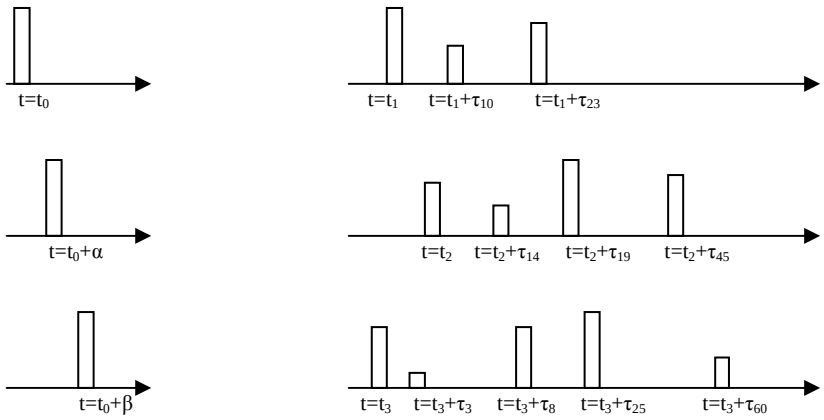


Figure 3.8: Time Variant Multi-Path Channel Response (α, β and τ represent time delays)

The variations in this type of channel are unpredictable and therefore have to be defined statistically. Consider a signal transmitted over a mobile communications channel, generally represented as:

$$s(t) = Re\left[s_l(t) e^{j2\pi f_c t} \right] \tag{3.20}$$

Where f_c is the carrier frequency and $s_l(t)$ is the information signal.
Assuming multiple propagation paths with their own propagation delay and attenuation, both of which are time variant, the received band pass signal can be represented thus:

$$x(t) = \sum_n \alpha_n(t) s(t - \tau_n(t)) \tag{3.21}$$

Where α_n is the attenuation factor of the signal and τ_n is the propagation delay. Substituting the transmitted signal into the received signal yields:

$$x(t) = Re\left\{\left[\sum_n \alpha_n(t) e^{-j2\pi f_c \tau_n(t)} s_l(t - \tau_n(t))\right] e^{j2\pi f_c t}\right\} \tag{3.22}$$

The equivalent low-pass version of the received signal is:

$$r_l(t) = \sum_n \alpha_n(t) e^{-j2\pi f_c \tau_n(t)} s_l(t - \tau_n(t)) \tag{3.23}$$

Now that the transmitted signal and received signal are in low-pass form, the low-pass channel can be described by the time-variant impulse response:

$$c(\tau; t) = \sum_n \alpha_n e^{-j2\pi f_c \tau_n(t)} \delta(\tau - \tau_n(t)) \tag{3.24}$$

This is the appropriate representation for a channel that contains discrete multi-path components rather than a channel that has continuous multi-path (for example channels which take into account tropospheric scatter).

Considering an un-modulated carrier signal with frequency f_c. The transmitted signal is therefore $s_l(t) = 1$ for all t and the received signal would be represented as:

$$r_l(t) = \sum_n \alpha_n(t) e^{-j2\pi f_c \tau_n(t)} = \sum_n \alpha_n(t) e^{-j\theta_n(t)} \tag{3.25}$$

This shows that the received signal is made up of the sum of a collection of time-variant vectors with amplitudes $\alpha_n(t)$ and phases $\theta_n(t)$. It can be seen from (3.25) that for $\alpha_n(t)$ to notably affect the received signal, large changes would have to occur in the medium. Whereas, the phase $\theta_n(t)$ will change by 2π radians with a change in τ_n of only $1/f_c$.

The delays $\tau_n(t)$ will change randomly and at different rates. Therefore the received signal is a random process and should be modelled as such. If the number of paths is large, the central limit theorem is valid and $r_l(t)$ can be modelled as a complex Gaussian random process and therefore the impulse response $c(\tau;t)$ is a complex random process with respect to t. As mentioned above, variations in the received signals, particularly in the phases $\theta_n(t)$, cause the signal to be altered. Sometimes the

combination of the signals at the receiver is constructive, increasing the amplitude of the signal, and at others, the combination is destructive, resulting in small or no amplitudes. This is signal fading. If the impulse response $c(\tau;t)$ is modelled as a zero-mean complex Gaussian process, the envelope $|c(\tau;t)|$ at any time instant t, is Rayleigh distributed and the channel is then referred to as a Rayleigh fading channel. If there are fixed scatterers or a line of sight (LOS) signal, the impulse response cannot be said to have zero mean and the envelope has a Rice distribution. In this case, the channel is referred to as a Ricean fading channel.

3.5.2 Fading Characteristics

The multi-path channel is time varying and so are its effects. If $s_l(t)$ is the transmitted signal, let $S_l(f)$ be the frequency content. Ignoring the effects of added noise, the received signal can then be represented in the time domain as:

$$r_l(t) = \int_{-\infty}^{\infty} c(\tau;t)s_l(t-\tau)d\tau \tag{3.26}$$

And in the frequency domain as:

$$r_l(t) = \int_{-\infty}^{\infty} C(f;t)S_l(f)e^{j2\pi ft}df \tag{3.27}$$

Where $C(f;t)$ is the Fourier transform of $c(t-\tau)$ and $S_l(f)$ is the frequency content of the transmitted signal $s_l(t)$.

When transmitting digital information over a channel, the pulse $s_l(t)$ is modulated with rate $1/T_s$, where T_s is the symbol interval. Equation (3.27) shows that the signal $S_l(f)$ is distorted by the time variant channel. If the bandwidth W of the signal $S_l(f)$ is greater than the coherence bandwidth, $(\Delta f)_c$, of the channel, the signal is affected by different gains and shifts in phase across the band. This is referred to as frequency-selective. The distortions caused by the time variations in the channel are viewed as changes in the signal strength and it is this that merits the term fading. In short, the multi-path spread or the coherence bandwidth of the channel in relation to the signal bandwidth determines whether the distortion is frequency-selective and the time variations, caused by the coherence time $(\Delta t)_c$ or by the Doppler spread B_d, cause the signal fading.

The way in which the channel affects the transmitted signal changes with decisions made about the signals duration and bandwidth. For instance, if the signal duration T_s is chosen such that it is much larger than the multi-path, or delay, spread T_m, there will be very little Inter Symbol Interference (ISI). If the bandwidth of $s_l(t)$ is approximately inverse to the channel bandwidth W:

$$W \approx \frac{1}{T_s} \tag{3.28}$$

And the signal duration is larger than the multi-path spread, then:

$$W << \frac{1}{T_m} \approx (\Delta f)_c \tag{3.29}$$

Which means that the signal bandwidth is much smaller than the coherence bandwidth of the channel. This is referred to as a frequency non-selective channel, implying that all the frequency components in $S_l(f)$ suffer the same attenuation and phase distortion upon transmission through the channel. In other words, within the bandwidth of the signal, $S_l(f)$, $C(f;t)$, the transfer function of the channel, has a constant complex frequency value. As $S_l(f)$ has frequency content focussed in the area around $f = 0$, $C(f;t)$ becomes $C(0;t)$ and as a result the received signal in the frequency domain (3.27) becomes:

$$r_l(t) = C(0;t) \int_{-\infty}^{\infty} S_l(f) e^{j2\pi ft} df$$

$$r_1(t) = C(0;t) s_l(t) \tag{3.30}$$

Therefore, when the signal bandwidth W is much smaller than the channels coherence bandwidth $(\Delta f)_c$, the received signal is the transmitted signal multiplied by the complex valued random process $C(0;t)$, representing the time variations within the channel. Therefore, if $W << (\Delta f)_c$ the multi-path components are said to be un-resolvable.

For a frequency non-selective channel, the transfer function $C(0;t)$ is written:

$$C(0;t) = \alpha(t) e^{-j\phi(t)} \tag{3.31}$$

With $\alpha(t)$ defining the envelope and $\phi(t)$ being the phase of the equivalent low-pass channel. If $C(0;t)$ is modelled as a complex Gaussian random process with zero-mean, the envelope has a Rayleigh distribution for any fixed value t. The phase $\phi(t)$ is uniformly distributed over $(-\pi, \pi)$ and the speed of the fading on the frequency non-selective fading channel is controlled by the correlation function $\phi_c(\Delta t)$ or by the Doppler power spectrum $S_c(\lambda)$.

It is also possible to use the channel parameters $(\Delta t)_c$ and B_d to characterise the fading rapidity, if the signal bandwidth W is chosen such that it is much smaller than the coherence bandwidth $(\Delta f)_c$ and the symbol interval T_s is chosen to be much smaller than the coherence time $(\Delta t)_c$. If T_s is smaller than the coherence time of the channel, the channel attenuation and phase shift are fixed for the duration of

at least one signalling interval and therefore the channel conditions are said to be slow fading. Also, if the bandwidth of the channel $W \approx 1/T_s$, when the channel is frequency non-selective and exhibits slow fading, the implication is that the product of the multi-path spread T_m and the Doppler spread B_d, referred to as the spread factor of the channel, must be less than unity. When the spread factor is less than one, the channel is said to be underspread as opposed to overspread.

3.5.3 Tap-Delay Line Channels

One major characteristic of the mobile radio communications channel is the effect of Inter-Symbol Interference (ISI). This occurs when the symbol interval is not sufficiently larger than the multi-path spread of the channel. One method of analysing a communications system under these conditions that is easy to work with is the tap-delay line model. Figure 3.9 gives an example of this type of channel. As the transmitter is broadcasting discrete-time symbols, with rate $1/T_s$ samples per second, and the receiver output is also at a rate of $1/T_s$ symbols per second, the channel can be represented by an equivalent transversal filter with tap gain coefficients

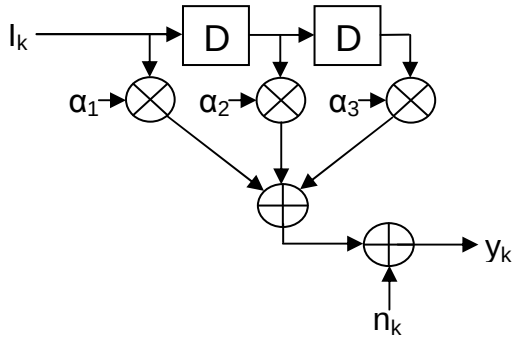


Figure 3.9: Tap-Delay Line Model of Channel with ISI

This research examines the effect of one time invariant discrete time model and a group of time varying channels on the BER and FER of short frame turbo codes. The first, time invariant model, consisted of three taps, each delayed from the previous by one symbol interval, representing the path loss of each symbol at the receiver. The tap-weights were [1 0.7499 0.5623]. Unfortunately the origin of these values cannot be given.

The second channel model group were developed from the UMTS models [UMTS1] and were intended to represent the effects of three environments, indoor office, pedestrian and vehicular. The specific channels are shown in table 3.1 below.

Indoor Channel A			Pedestrian Channel A			Vehicular Channel A		
Tap	Power (dB)	Delay (ns)	Tap	Power (dB)	Delay (ns)	Tap	Power (dB)	Delay (ns)
1	0	0	1	0	0	1	0	0
2	-3	50	2	-9.7	110	2	-1	310
3	-10	110	3	-19.2	190	3	-9	710
4	-18	170	4	-22.8	410	4	-10	1090
5	-26	290				5	-15	1730
6	-32	310				6	-20	2510

Table 3.1: UMTS Channel Specifications

The data rate chosen was 480 kbps and the carrier frequency was set at 2GHz. The data rate is also the symbol rate as BPSK modulation was used as before. Each BPSK symbol was represented by 8 samples, selected as a good compromise between the suppression of aliasing errors and the time required by simulation [JER84]. This implies that the duration of one sample is:

$$\frac{1}{480ks/s \times 8} = 260.417ns$$

This symbol sample duration corresponds to the tap delays of each individual channel model. As an example, considering the indoor channel, it can be seen that the first four taps all act upon the same sample.

For the indoor channel, the Doppler shift was approximated to zero. However for the two mobile channels, the time varying Rayleigh distributed tap coefficients were filtered according to the Doppler frequency response required.

The classic mobile radio channel Power Spectral Density (PSD), detailed in [JAK74] is:

$$PSD = \begin{cases} \frac{E_0}{4\pi f_m} \frac{1}{\sqrt{1-(\frac{f}{f_m})^2}} & |f| \le f_m \\ 0 & elsewhere \end{cases} \quad (3.32)$$

Where E_0 = Energy constant

The problem with this power spectral density representation is the step to infinity at $f_c \pm f_m$. This is shown in figure 3.10.

Aulin describes another representation including the angle of signal arrival (β_m). This is:

$$\text{PSD} = \begin{cases} 0 & |f| > f_m \\ \dfrac{E_0}{4\sin\beta_m}\dfrac{1}{f_m} & f_m \cos\beta_m \le |f| \le f_m \\ \dfrac{1}{f_m}\left[\dfrac{\pi}{2} - \arcsin\dfrac{2\cos^2\beta_m - 1 - (f/f_m)^2}{1-(f/f_m)^2}\right] & |f| < f_m \cos\beta_m \end{cases} \quad (3.33)$$

However, this representation has sharp discontinuities at $\pm\beta_m$ and therefore the more complex, but more realistic PSD as proposed by Parson in [PAR00] is used. Here, the PDF of the angle of arrival $p_\beta(\beta)$, is represented as:

$$p_\beta(\beta) = \begin{cases} \dfrac{\pi}{4|\beta_m|}\cos\left(\dfrac{\pi}{2}\cdot\dfrac{\beta}{\beta_m}\right) & |\beta| \le |\beta_m| \le \pi/2 \\ 0 & \textit{elsewhere} \end{cases} \quad (3.34)$$

This is then applied to the following equation to arrive at the power spectral density:

$$PSD = FT\left[\frac{E_0}{2}\int_{-\pi}^{\pi} J_0(2\pi f_m \tau \cos\beta)\, p_\beta(\beta)\, d\beta\right] \quad (3.35)$$

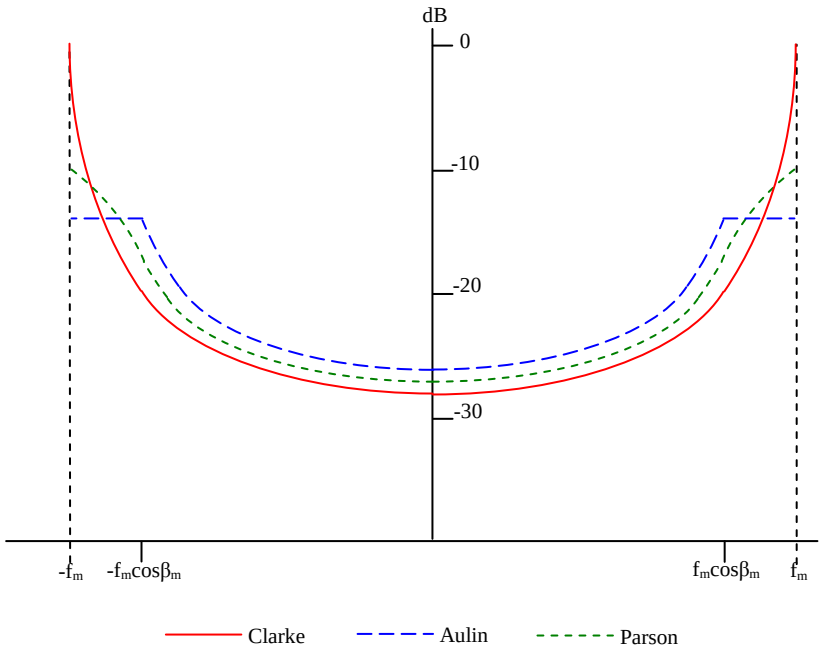


Figure 3.10: Power Spectral Density Models

3.6 Conclusion

This chapter has served as a background for forthcoming chapters, describing the modulation technique that will be used as well as the channels that will be encountered.
The goal of error control coding is to counteract the perturbations encountered within the transmission medium. To understand the methods used in neutralising these effects, a comprehension of how and why they occur is first necessary.

The chapter begins with a description of BPSK modulation, the method used here to connect the digitally encoded information to the communications channel. A mathematical description of the modulator helps to show how the signal can be represented for analysis of the error control code.
Following this, the channel models that describe the behaviour of transmitted signals in mobile radio communications systems were presented, beginning with one of the simplest analysis tools available to the communications system designer, Additive White Gaussian Noise. A description of the Gaussian process and how this is tailored for use in communications analysis was given as well as a description of the Rayleigh process, also commonly encountered in communications systems and derived from the Gaussian.

Although AWGN is a commonly used analysis tool, it is by no means a credible model for a mobile communications channel. A more definitive description of the mobile communications channel was therefore made. The causes and effects of large and small scale fading were explained followed by the multipath effect. The alterations that can occur in the time and frequency domains were described and the ways in which they control the properties of the channel were explained.

Finally, a method for representing this type of channel was described for analysis of the error control coding schemes that will be described later.

Chapter 4

Theoretical Background Of Turbo Codes

4.1 Introduction

The aim of this chapter is to present a mathematical framework for turbo codes. The chapter begins by introducing the basic theoretical principles of error control coding, starting with the concept of redundancy and outlining its effect on the error control capabilities of a code.

To give a clear understanding of the code and its properties, more conventional coding schemes are referred to and covered more extensively in the appendices.

Following this short introduction, each of the component processes involved in the turbo-coding scheme is examined in detail.

The application of convolutional codes, as components in the turbo encoder system, is then studied, specifically the variations in design from conventional codes and the reasoning behind them. The methods involved in effectively 'tail biting' these component encoders where necessary, in preparation for their use in the turbo encoder, are then examined. This is followed by an explanation of the interleaving mechanism, its function, importance and design, and a discussion on the puncturing mechanism strategy.

Following the analysis of the design of the encoder, an overview of the turbo decoder system in general is given, highlighting the manipulation of the received data that is necessary prior to decoding. Finally, mathematical analysis of the two common soft-input soft-output decoding algorithms, SOVA and MAP, and their function in the turbo decoder system is presented. Further to this, variations on the MAP algorithm, namely the Max-Log-MAP and Log-MAP algorithms, are developed and rationalised.

4.2 Error Control Coding

As explained in chapter 3, when communicating over any kind of channel, the signal will always suffer distortion.

Depending on the application, there are various ways to counteract this distortion, or at least lessen its effects. Error control coding is one method. Its application is in the transmission of any type of digital information. This information could be derived from voice, text or pictures. The concept is simple, the process can be as straightforward or as complicated as necessary, but it is all a development of the concept of redundancy, described in appendix A.

The use of redundancy allows codes to detect and possibly correct errors. However, it also means a reduction in the code rate as explained in appendix B.

To detect the errors in received codewords, the redundant parity bits must be added in a manner that is understood by the receiver. Appendix C explains modulo-*q* arithmetic, which is used to satisfy this need.

Appendix D shows how the error control capabilities of a code can be found through calculating the Hamming distance of a code.

Having covered the basics, appendix E explains one of the most common coding schemes. Convolutional codes are also one of the main building blocks of the turbo code system.

Appendix F shows how the Hamming distance of these codes is calculated. Although it is possible to apply the methods explained in appendix D, the fact that convolutional codewords are semi-infinite means that using this method is not practical.

4.3 Decoding Convolutional Codes – The Viterbi Algorithm

Convolutional codes are one of the main components of turbo codes. To aid in the explanation of the turbo decoding methods, the conventional convolutional decoder is also described here.

The Viterbi algorithm [VIT67] is a Maximum Likelihood Sequence Estimator (MLSE) that searches through a trellis and finds the most likely codeword. The algorithm compares the received symbol at time *t* with all possible transitions in the trellis, keeping a record of the paths that are closest to the codeword in terms of Hamming distance as it progresses. As with all the great inventions and discoveries, the algorithm is simple once it has been explained. It can be expressed as follows:

1. Begin at time $t = 0$, at state 00.
2. Calculate the distance between the first received symbol and the symbol associated with each trellis path at current time.
3. Select the path with the lowest distance at each state node and store both the path and the distance.
4. If there is more than one path with the lowest distance, select one arbitrarily.
5. Increment time t and return to step 2. Add the lowest distance to the previous lowest distance and store with the sequence of paths. This running score is called the survivor score and the corresponding sequence of paths the survivor path.
6. Continue this process until all the symbols in the codeword have been compared.
7. The resulting survivor path with the lowest survivor score is the most likely transmitted codeword.

An example of this algorithm is given below for the example $[7;5]_8$ non-systematic convolutional code (note the error in the received codeword):

Transmitted codeword	:	00 11 10 00 10
Received codeword	:	00 1**0** 10 00 10
Error vector	:	00 01 00 00 00

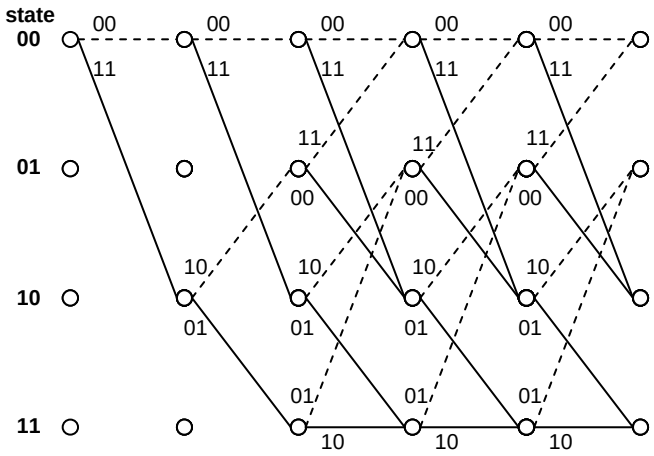


Figure 4.1: Trellis representation of $[7;5]_8$ non-Systematic Convolutional Code

The following table shows the process at each time step with the survivor score or scores highlighted in red. As each received pair (or symbol) is received, it can be compared with the possible transitions within the trellis at that step in time. The final column of the table below shows the survivor path according to the algorithm. Note that at the second time step, where an error has occurred in transmission, there are two possible paths.

Received Pair	Possible Transition Pairs	Transition Scores	Survivor Path
00	00 11	0 2	S_0S_0
10	00 11 10 01	1 1 2 4	$S_0S_0S_2$ ***or*** $S_0S_0S_0$
10	00 11 11 00 10 01 01 10	2 2 3 3 1 3 6 4	$S_0S_0S_2S_1$
00	00 11 11 00 10 01 01 10	2 5 3 1 3 4 5 5	$S_0S_0S_2S_1S_2$
10	00 11 11 00 10 01 01 10	3 3 4 4 1 3 6 4	$S_0S_0S_2S_1S_2S_1$

Table 4.1: Viterbi Algorithm Results When Applied to Received Sequence

The trellis below shows the final survivor path in bold type. The survivor scores for every path at each node at each time step are also shown, with the lowest score highlighted in bold italics.

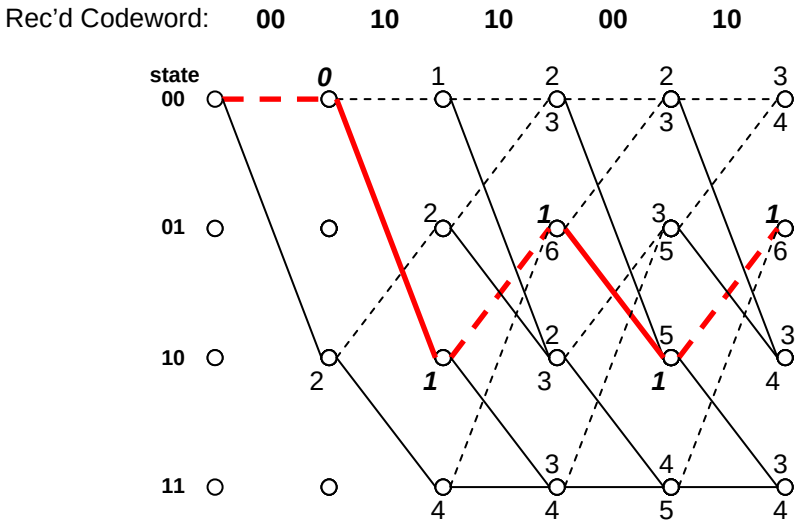


Figure 4.2: Survivor Path After Viterbi Decoding

4.4 Turbo Codes

Originally proposed by Berroux *et al* [BER93], Parallel Concatenated Convolutional Codes (PCCC) or turbo codes have quickly risen to the forefront of contemporary coding theory. The remarkable properties exhibited by these codes have given the coding community a fresh direction in its efforts to reach Shannon's channel capacity [SHA48].

Using multiple component encoders in parallel concatenation to help combat bursts of channel interference and multiple decoders in iterative, serial concatenation to allow useful information to be passed from one decoder to the next, these revolutionary codes are producing results closer than ever to the theoretical limit.

Reaching, or at least approaching, the theoretical channel capacity limit is in itself a huge achievement. But when this is combined with the turbo coding schemes relatively low complexity, it is possible to see their potential as a possible solution to the ever increasing demands made on today's mobile communications networks.

4.4.1 The Turbo Encoder

In general terms, the turbo encoder consists of *n* (where *n>1*) component convolutional encoders, each separated from the one before by an interleaver. Generally these encoders are identical, although this is not essential. The component encoders must be recursive-systematic however, for a variety of reasons examined later.

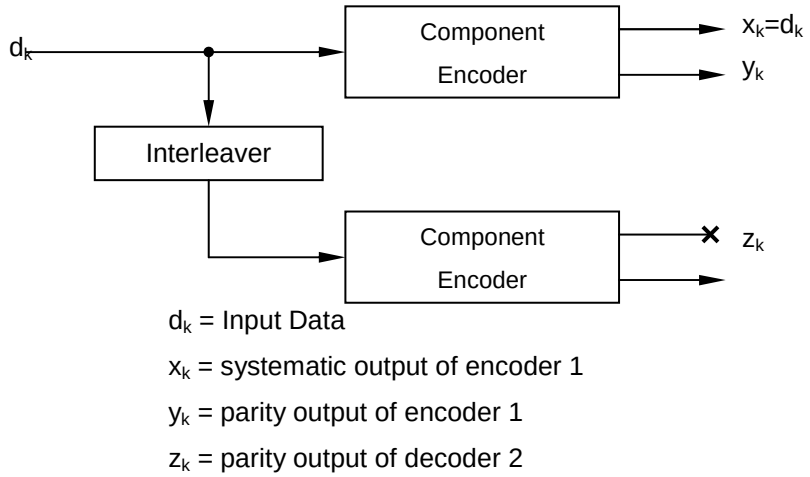


Figure 4.3: Basic Turbo Encoder

Figure 4.3 above, shows a very general view of a turbo encoder. Two recursive systematic convolutional encoders are separated by an interleaver equal in size to the dataword input to the system. The fact that the encoders are systematic implies that it is unnecessary to transmit systematic information from any but the first, as the interleaver structure is known at the decoder.

4.4.2 Component Encoders

Recursive systematic convolutional encoders are derived directly from the non-systematic version. Figure 4.4 is an example of a $[7;5]_8$ recursive systematic convolutional encoder with the same free distance and trellis structure as the non-systematic convolutional encoder in appendix E. Note that the output bit sequences for this encoder differ from those of the non-systematic equivalent.

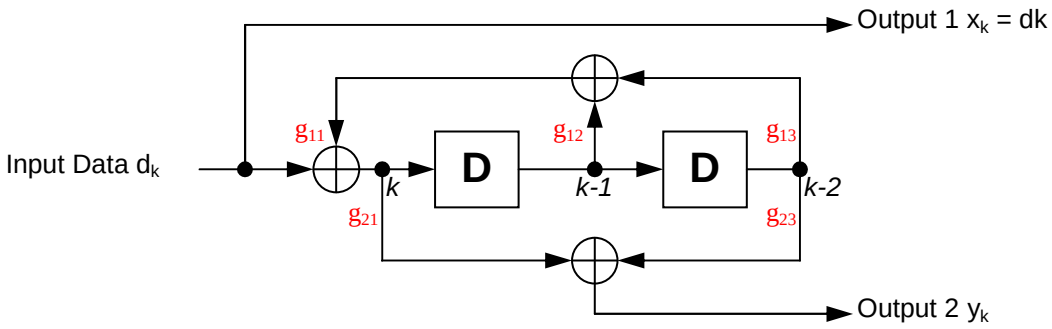


Figure 4.4: $[7;5]_8$ Recursive Systematic Convolutional Encoder

The recursive systematic convolutional encoder is derived from the non-systematic version by dividing through by the first generator vector, thus the generator matrix becomes $\left[1, \frac{g_2}{g_1}\right]$, instead of $[g_1, g_2]$. This effectively adds a feedback loop.

As an example, consider the example non-systematic encoder of appendix E. With rate R = 1/2, constraint length K, memory $m = K - 1$. The input, at time k, is d_k and the corresponding output is (x_k, y_k). x_k and y_k are defined as:

$$x_k = \sum_{i=0}^{m} g_{1i} d_{k-i} \text{ , where } g_{1i} = 0 \text{ or } 1 \tag{4.1a}$$

$$y_k = \sum_{i=0}^{m} g_{2i} d_{k-i} \text{ , where } g_{2i} = 0 \text{ or } 1 \tag{4.1b}$$

Setting the output $x_k = d_k$ and adding a feedback loop whose components are g_1 and whose output (a_k) becomes the input to the shift registers gives:

$$x_k = d_k \tag{4.2}$$

$$a_k = d_k + \sum_{i=1}^{v} g_{1i} a_{k-i} \tag{4.3}$$

$$y_k = \sum_{i=0}^{v} g_{2i} a_{k-i} \tag{4.4}$$

Berrou *et al* [BER93] show that a_k in the recursive systematic encoder displays the same statistical properties as d_k in the non-systematic decoder, that is that they both take on values of either 0 or 1 with equal probability.

Figure 4.5 shows the trellis representation of the example recursive systematic encoder of figure 4.4. As can be seen, when comparing this trellis with the trellis of the equivalent non-systematic convolutional encoder, the structure remains unchanged, as does the minimum weight d_{min} or d_{free}. The mapping of inputs-to-outputs are all that has changed.

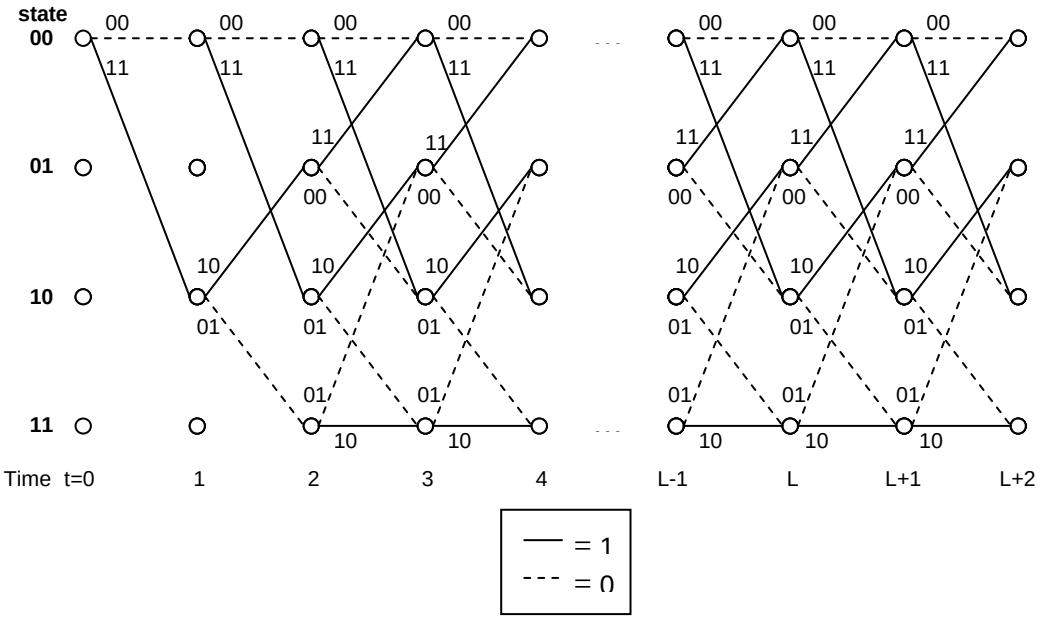


Figure 4.5: Trellis representation of $[7;5]_8$ Recursive Systematic Convolutional Encoder

The use of systematic encoders means that the rate of the turbo encoder system can be reduced as only one of the component encoders need send the systematic bits. For instance, figure 4.3 shows a turbo encoder system comprised of two rate 1/2 component convolutional encoders. As indicated by figure 4.4, the first output of a recursive systematic encoder follows the input. In the turbo encoder system, the systematic output of the first component encoder is equal to the input of the system. The two component encoders are separated by an interleaver and thus the systematic output of the second encoder is an interleaved version of the same turbo system input. The same interleaver process is used by the decoder so it is unnecessary to transmit the systematic bit stream of the second encoder as this can be obtained from that of the first encoder. Because non-systematic encoders do not, by definition, produce an exact replica of the input as part of their output, it is not possible to omit part of the second encoder's output.

However, this reduction in rate is secondary, more important are the weight properties of the turbo encoder. This is the main reason that Berroux *et al* proposed the recursive-systematic format. Consider the non-recursive encoder applied to the turbo encoder system. A weight-1 input (000…010) will produce a low weight codeword, equal to that of the generator matrix (a string of zeros will lead to the encoder returning to, and remaining at, the all-zero state, producing weight-0 outputs). The interleaved input will still be weight-1, so the output of the second encoder will have the same weight.

A recursive systematic encoder on the other hand, does not return to the all zero state after a weight-1 input. This means that, as the length of the data input to the encoder increases, the weight of the codeword does too, even though the weight of the data word may not. The implication is therefore that the weight of a codeword produced by an infinitely long weight-1 data word would be infinite. In reality, the length of the data word is limited and the weight of the codeword is therefore sometimes referred to as semi-infinite. As mentioned earlier, with a weight-1 input, there is no pattern for the interleaver to affect and as a result, the second component encoder will produce another semi-infinite codeword.

With the recursive encoder, it is an input equal to g_1 that will give the lowest weight output, equal to that produced by a weight-1 input to a non-recursive encoder. Considering just such a weight-3 input word, equal to g_1 for the example encoder of figure 4.4 (000…111000), it can be seen that the output codeword of the first component encoder does indeed equal that of the weight-1 input to the non-systematic convolutional encoder. Interleaving this, such that the dataword is not divisible by g_1 (001…011000) again yields a semi-infinite output from the second encoder. Applying the same two datawords to the non-systematic encoder produces a finite weight codeword for both.

These properties help to improve the over all weight of the turbo codeword. Taking the worst-case scenario for a turbo encoder constructed with non-systematic $[7;5]_8$ convolutional codes (a weight-1 input), both component codewords are finite and therefore the turbo codeword is finite. Applying the same criteria to the recursive-systematic equivalent turbo encoder (worst case scenario, input divisible by g_1), the weight of the first output codeword will be finite, however, with a well designed interleaver process, the output of the second encoder will be semi-infinite and therefore the output of the turbo encoder will itself be semi-infinite. As discussed above, the minimum codeword weight of a code determines its error controlling capabilities, therefore a code with a semi-infinite minimum codeword weight will far outperform one without.

4.4.3 Tail Bits

As discussed in chapter 2, to simplify and improve the decoding process, the input dataword to the turbo encoder is tailored so that the codeword from the first component encoder (and sometimes the second) finishes at the all-zero state. The bits appended to the dataword are referred to as tail bits. For example, assume a five-bit dataword, ‘01101’, arrives at the first component encoder. The trellis path described by the codeword created from this input is highlighted in red in figure 4.6, finishing at state ‘01’. Note that the trellis has been extended in time by two steps. The only paths shown in this extension are those that represent the return from any state back to the all-zero state. Returning from state ‘01’ to the all-zero state can be accomplished by adding a single bit to the input dataword, as illustrated by the dashed blue line in the same figure. Observe, however, that to return from state ‘10’ or state ‘11’ to the all-zero state would require the addition of two bits and that no bits are necessary if the original dataword leads to a codeword terminating at the all-zero state.

To maintain a uniform frame size, the methods adopted in this thesis will always append the maximum necessary number of bits to the dataword, in this case 2 bits per frame, demonstrated by the bold green lines in figure 4.6.

Obviously, the tail bits must be ignored once the decoding process has been completed, as they are not part of the original data. Various techniques have been applied to component encoders to improve the decoding process, these can be broken down into those that are based on a 'look-up' table approach and those that are incorporated in the decoder.

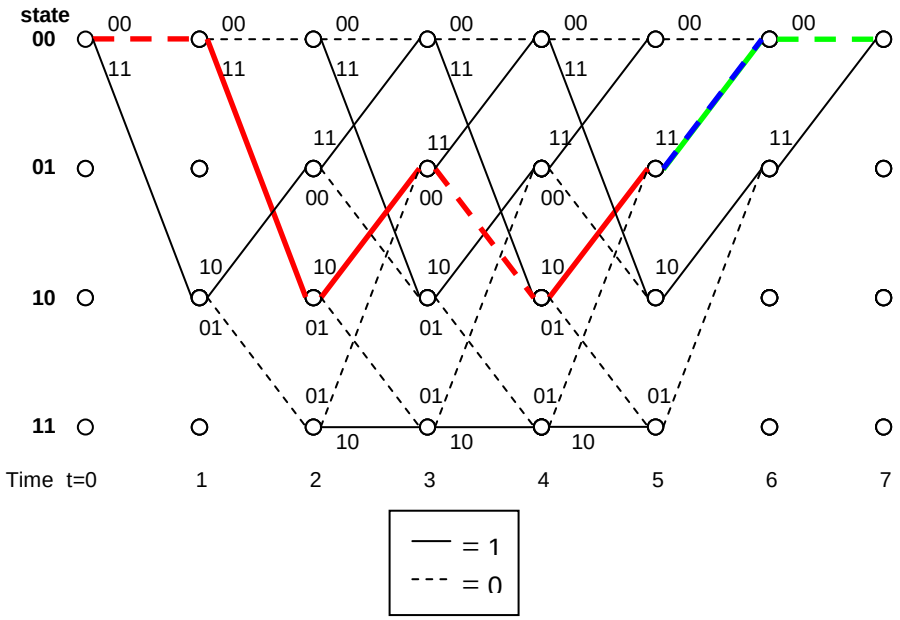


Figure 4.6: Example of Techniques for Adding Tail bits

The 'look-up' table approach simply encodes the dataword input and then determines the state at which the codeword has terminated. There are two options then available to the code designer. The first option is to append the least bits necessary to return the trellis of the codeword to the all-zero state. This method is not practical for a real world application, as it would require the transmission of either each frame length or the number of tail bits attached. The second method, as mentioned above, is to append a fixed number of bits, equal to the maximum number of tail bits necessary to return any state to the all-zero state. In the example $[7;5]_8$ code, this is 2 tail bits per frame.

The second technique used to effectively 'tail' a codeword was first proposed by Divsalar and Pollara [DIV95] and involves the addition of a simple switch to both encoders in order that they are both forced to return to state S0 (see figure 4.7 below).

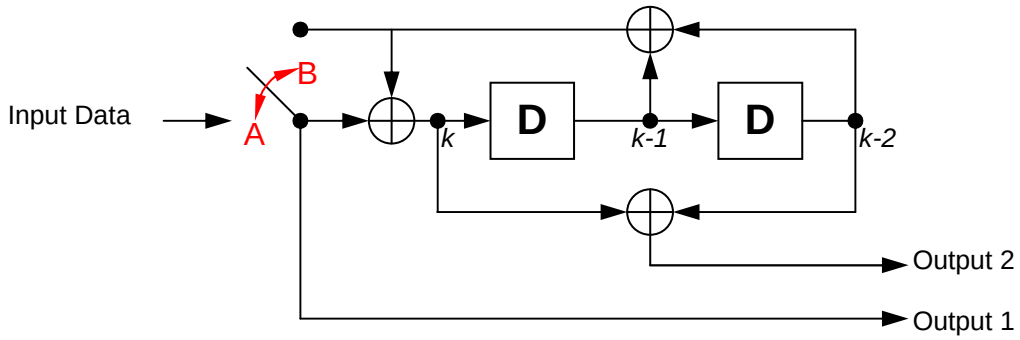


Figure 4.7: Example of Divsalar - Pollara Tailing Technique

Encoding commences with the switch in position A. Once the dataword has been read the switch moves to position B, causing the output of the feedback loop to become the input to the encoding system. This has the same effect as the look-up table, requiring a maximum of two encoder operation sets to return from any state to the all-zero state (for the example $[7;5]_8$ code) and may offer a more practical solution for hardware implementations or where memory is at a premium. It should be noted that in practice, the longest straight return path to the all-zero state for any given convolutional encoder is equal to the constraint length, K, – 1.

4.4.4 The Interleaver

The role of the interleaver in a turbo code is two-fold. Traditionally interleaving has been applied to the codeword as a method of reducing the effect of bursts of errors that might be longer than those effectively combated by the code. The channel interleaver ‘scrambles’ the bits of the codeword using a method that is known by the decoder, thus any long bursts of errors endured by the transmitted data stream are broken up on reception by the de-interleaver at the other end of the channel. With regard to turbo codes specifically, the iterative nature of the decoding system means that, although the information passed to the first component decoder on reception is not interleaved, the information received by the second component decoder is. The fact that the two information sequences are de-correlated means that elements of the data stream that remain improperly decoded by one decoder stand a better chance of being decoded at the second

While this is not the primary role of the interleaver in a turbo encoder system (the interleaver is situated between the two component encoders rather than after the encoding process), it is still a useful bi-product.

The principal purpose of the interleaver here is to increase the weight of the output codeword. It is important to recognise that the interleaver does not alter the weight of the input dataword, instead rearranging the bits contained within. Thus a dataword that produces a low-weight codeword might produce one with higher weight if the bits are rearranged. In this way, the likelihood of low-weight turbo codes is reduced. Table 4.2 gives an example of three different permutations of a five-bit, weight-2 dataword and the difference in codeword weights for the recursive systematic $[7;5]_8$ example code.

Dataword	Codeword	Codeword Weight
01100	0011100001	4
01010	0011011001	5
10010	1101011100	6

Table 4.2: Dataword Permutations versus Codeword Weights

For optimal performance, the interleaver in a turbo encoder should be tailored to the weight distribution of the input data, the effect of the channel and the size of the frame. However, most of this would mean constant re-evaluation of parameters and changes in interleaver structure, which would mean that each new structure would have to be transmitted to the decoder, blocking valuable bandwidth. Therefore compromises must be made and for this reason most of the time, especially for comparison simulations, a pseudo random interleaver is used, sometimes tailored to both frame length and prevalent dataword weights or solely frame length.

Although the difference in performance between one pseudo-random interleaver of sufficiently large dimension and another is negligible [RYA99], designers still search for the 'optimal' design for each new code. The search for this interleaver is endless, thus Barbulescu and Pietrobon [BAR94] have suggested an acceptable search system. The following algorithm describes this method.

1. Generate a random interleaver.
2. Apply all possible input data sequences.
3. For each of these sequences determine the output codeword
4. Calculate these codeword weights.
5. Determine the weight distribution of the code.
6. Find the minimum codeword weight and the number of codewords with this weight

Repeat steps 1-6 for a practical number of times and select the interleaver with the largest minimum codeword weight and the lowest number of codewords with that weight.

In reality, this is not a particularly practical system. For each framelength there are a very high number of possible interleavers and it is for this reason that the search is ended before all possible candidates have been investigated. This means that, unfortunately, 'a practical number of times' becomes 'a practical length of time' and as such a greater proportion of possible interleavers tend to be considered for smaller frame lengths than for large.

Note also that if all possible datawords are known, tailoring the design of the interleaver can be beneficial. As described in 4.4.2, the lowest weight component codewords are those obtained from a dataword that is divisible by the first generator, g_1. Therefore, if all data inputs are known, the interleaver can be designed such that the input to the second encoder is never divisible by that generator, increasing the minimum turbo codeword weight and, as a result, improving the codes error control capabilities.

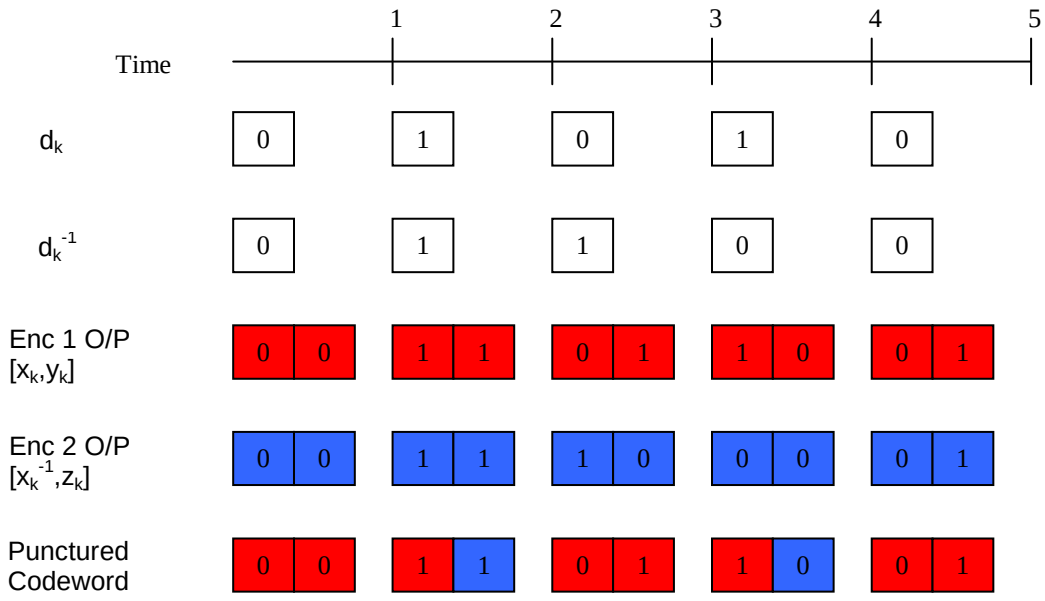
4.4.5 The Puncturing Mechanism

Now more than ever, bandwidth is at a premium, at least for mobile communications channels (less so for satellite channels). It is therefore necessary to design codes that are as efficient as possible. Often concessions must be made regarding performance and bandwidth. Information that is duplicated must, of course, be removed before transmission, hence the omission in the turbo encoder of the systematic bits from the second component encoder which can be obtained at the decoder end of the system. Puncturing is used to further reduce the length of the codeword frame length.

Without puncturing, the length of the turbo codeword, using rate-1/2 component codes, would be three times that of the dataword (one systematic and two parity bits per data bit). Puncturing reduces the number of bits in the codeword and therefore the bandwidth necessary for transmission. Unfortunately, puncturing also reduces the code performance, so some compromise is necessary.

There are various puncturing mechanisms that can be applied to the output of the turbo encoder ([HAG96], [CHA05]). The system described in figure 4.8, has been used in [HAG96] amongst others and is the system used wherever puncturing was required in this body of research.

The system is simple, each systematic bit from the first component encoder is accompanied by one parity bit, alternating at each time step, between the two parity bit output streams.



d_k = Input Data

d_k^{-1} = Interleaved Input Data

x_k = systematic output of encoder 1

x_k^{-1} = systematic output of encoder 2

y_k = parity output of encoder 1

z_k = parity output of encoder 2

Figure 4.8: Puncturing Technique

4.4.6 Effective Free Distance

As explained earlier, the Hamming distance of a convolutional code gives an indication of the error control properties of that code. The nature of turbo codes is such that the output of second component encoder will preferably not have the same weight as that of the first and without knowing this output, it is not practical to calculate the Hamming distance of a turbo code. There is also the fact that, unlike conventional convolutional codes, the recursive systematic variety will never return to the all-zero state with a weight-1 input.

[BEN96a], amongst others, developed the 'effective free distance', referred to as $d_{free,eff}$, of turbo codes which can be regarded as the equivalent of the Hamming distance. As with Hamming distances, the calculations begin with the computation of the lowest weight recursive systematic codeword created by a weight-2 input dataword. The weight of the parity bits in this codeword is called z_{min} and it is this that is used to find the lowest weight of the turbo code:

$$d_{free,eff} = 2 + (2 \times z_{min}) \tag{4.5}$$

To determine whether the chosen component code is optimal for it's chosen parameters it is therefore necessary to be able to calculate the maximum possible z_{min}. Fortunately the authors of the same paper theorised and proved that this is a function of the rate of the component code and the number of memory elements:

$$z_{min,optimal} = (n-1)(2^{v-1} + 2) \tag{4.6}$$

Where v is the number of memory elements of the component code and n corresponds to the rate $1/n$ of the code.

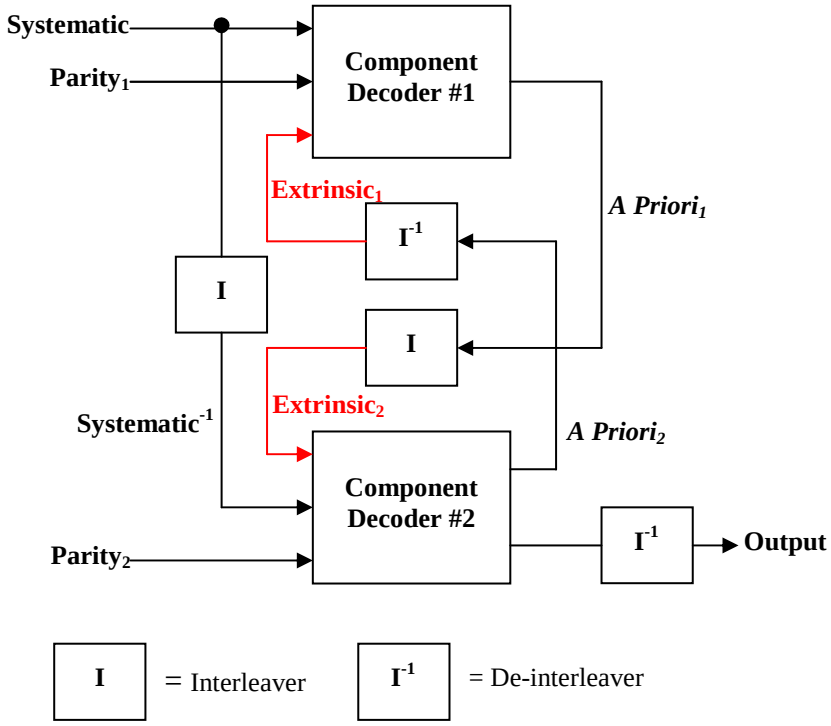


Figure 4.9: Turbo Decoder Construction

4.5 Turbo Decoding

The turbo decoder is constructed of Soft-Input Soft-Output (SISO) decoders in serial concatenation as opposed to the parallel concatenation of the encoder. The number of component decoders is equal to the number of component encoders used in the turbo encoder.

The decoding strategy and remarkable results produced by turbo codes hinge on the use of these SISO decoders. Unlike conventional decoders, for instance the Viterbi decoder, the component decoders used in turbo codes do not take binary values as their inputs, rather the received values from the demodulator, appropriately scaled with respect to the channel amplitude, hence the term 'Soft-Input'.

The component decoder outputs are also soft. Both input and output follow the same form, that is, a non-binary value, the sign of which indicates the hard decision value and the magnitude of which gives an idea of the reliability of the hard decision.

Before decoding can begin, the received data must be sorted into the separate systematic and parity sequences. Any erasures must then be inserted to match the original data frame length.

Figure 4.9 shows a generic turbo decoder. By examining this diagram, the turbo decoder system can be followed. Before decoding commences, the received codeword frame is split into the three sequences output from the turbo encoder prior to their combination into a single frame, the original systematic (or data) and two parity sequences.

Each component decoder takes as its inputs a systematic sequence and a parity sequence together with any available extrinsic information. This information is derived from the output of the previous component decoder and is applied as *a priori* information, acting as a further indication of the reliability of the received data stream.

At the beginning of the first turbo decoder iteration of a received codeword frame, there is no previous extrinsic information, therefore only the systematic and parity from the first encoder (**Parity$_1$**) are applied to the first component decoder. The output of this component decoder contains *a posteriori* information. From this, *a priori* information, to be used by the second component decoder is extracted by subtracting the systematic input and its matched *a priori* input (**Extrinsic$_1$**).

Before the second component decoder can begin, the inputs must be arranged correctly. The second parity sequence (**Parity$_2$**) was originally created at the turbo encoder by using an interleaved version of the original data stream, therefore the received systematic must be appropriately interleaved to match (**Parity$_2$**), becoming (**Systematic^{-1}**). The *a priori* information gained from the first component decoder must also be interleaved to match the sequence of data input to the second component decoder. Thus, the inputs to the second component decoder are the interleaved systematic (**Systematic^{-1}**), the second parity stream (**Parity$_2$**) and the interleaved *a priori* stream (**Extrinsic$_2$**).

Once the process at the second component decoder is complete, any extrinsic information must be obtained for use by the first component decoder at the beginning of the next turbo iteration. Therefore, the interleaved systematic (**Systematic^{-1}**) and the *a priori* information (**Extrinsic$_2$**) are subtracted from the output to find the *a posteriori* information produced by the component decoder.

This brings the turbo decoder back to the first component decoder. This time, there is *a priori* information available. It must first, however, be de-interleaved to match it to the original data sequence, becoming (**Extrinsic$_1$**). Along with the original data sequence and the first parity sequence (**Parity$_1$**), this completes the inputs to the first component decoder for the second iteration and the process described above begins again.

The turbo decoder will repeat this process until a sufficient number of iterations have been completed. The point at which it stops decoding a particular frame can be determined in a number of ways. The duration of the decode could be a stopping factor, with the turbo decoder having a fixed time allowed for each code frame received. In the same way, a fixed number of iterations could be specified. A third method is to examine the change in *a priori* values. As the decoder repeats the decoding sequence, the increase in these reliability values will reduce, indicating that more iterations will create only a small improvement in decoded output. When a particular threshold for the rate of reliability information increase is reached, the decoding of that codeword is terminated.

Once all iterations are complete, hard decisions are made on the de-interleaved output of the second component decoder.

4.5.1 Component Decoders

There are two main decoding strategies used by turbo codes, the Soft Output Viterbi Algorithm (SOVA) and the Maximum *A Posteriori* (MAP) algorithm. The latter was proposed in the original turbo code publication [BER93] and the former was first proposed in [HAG89] and for use in the turbo system in [BER93a].

The benefits of the SISO decoding algorithms and their use of *a priori* reliability information becomes obvious when considering the possibility of two competing paths with the same accumulated metric. When using the original Viterbi algorithm, there is no distinction between these two paths and therefore, according to the algorithm, they both have equal likelihood of being the correct one. As the algorithm stands, there is no provision for making a decision between them and as such, an implementation of the algorithm must arbitrarily decide which is the survivor and which is the competitor. Even for a simple binary code with only two transitions to and from each state it is obvious that the probability of choosing the incorrect path is equal to the probability that the chosen path is correct. SISO decoding algorithms avoid such arbitrary decisions.

The combination of the component decoders' abilities to take soft information as their inputs and to output information in the same manner really begins to make a difference when applied to the turbo decoder strategy. Although both component decoders are working to ascertain the same dataword, the inclusion of the interleaver means that where one codeword suffered excessive distortion and long bursts of errors during transmission, the second may not have. Therefore, the reliability information of one decoder can be used to repair the sections of the subsequent decoder codeword. This can be further refined by repeat decoding, hence the iterative nature of the turbo decoder structure.

4.5.2 Soft Output Viterbi Algorithm (SOVA) Decoding

The SOVA decoding algorithm is the less complex of the two popular turbo decoding algorithms. Based on the Viterbi algorithm [VIT67] and developed over the years, [FOR73] and many others, the earliest documentation proposing its use for concatenated codes is [HAG89]. Although less complex, SOVA has a degradation of about 0.7dB (at BER=10^{-4}, memory 4) [PAP96] in comparison with MAP.

The Log-likelihood algebra used by the SOVA has been covered extensively by Hagenauer [HAG96]. It is based on modulo-2 addition of the binary random variable u_k, which is -1 for logic 0, and +1 for logic 1. *L(u)* is the log-likelihood ratio for the binary random variable and is defined as:

$$L(u) = ln\frac{P(u=+1)}{P(u=-1)} \tag{4.7}$$

This is described as the 'soft' value of the binary random variable *u*. The sign of the value is the hard decision while the magnitude represents the reliability of this decision. As *L(u)* increases towards +∞, the probability that *u* = +1 also increases, and as *L(u)* increases towards -∞, the probability that *u* = -1 increases.

This probability can be re-written as a conditional probability on another random variable, in this case the received bit *y*, and becomes the conditional log-likelihood ratio *L(u|y)* defined as:

$$L(u \mid y) = ln\frac{P(u=+1 \mid y)}{P(u=-1 \mid y)} \tag{4.8}$$

The probability of the sum of two binary random variables can be found thus:

$$P(u_1 \oplus u_2 = +1) = P(u_1 = +1)P(u_2 = +1) + P(u_1 = -1)P(u_2 = -1) \tag{4.9}$$

Where the symbol ⊕ represents modulo-2 addition.

This equation can be rearranged, using the fact that the sum of all probabilities is equal to one, or, $P(u=-1) = 1 - P(u=+1)$. Therefore, equation (4.9) becomes:

$$P(u_1 \oplus u_2 = +1) = P(u_1 = +1)P(u_2 = +1) + (1 - P(u_1 = +1))(1 - P(u_2 = +1)) \tag{4.10}$$

Directly from (4.7):

$$e^{L(u)} = \frac{P(u=+1)}{P(u=-1)}$$

$$e^{L(u)} = \frac{P(u=+1)}{1-P(u=+1)}$$

$$e^{L(u)} - e^{L(u)}P(u=+1) = P(u=+1)$$

$$e^{L(u)} = (1+e^{L(u)})P(u=+1)$$

$$P(u=+1) = \frac{e^{L(u)}}{1+e^{L(u)}} \tag{4.11}$$

So for u_1:

$$P(u_1=+1) = \frac{e^{L(u_1)}}{1+e^{L(u_1)}} \text{ and } P(u_1=-1) = 1-\frac{e^{L(u_1)}}{1+e^{L(u_1)}} \tag{4.12}$$

and for u_2:

$$P(u_2=+1) = \frac{e^{L(u_2)}}{1+e^{L(u_2)}} \text{ and } P(u_2=-1) = 1-\frac{e^{L(u_2)}}{1+e^{L(u_2)}} \tag{4.13}$$

Using these equivalencies and substituting into equation (4.10), gives:

$$P(u_1 \oplus u_2 = +1) = \frac{e^{L(u_1)}}{1+e^{L(u_1)}} \cdot \frac{e^{L(u_2)}}{1+e^{L(u_2)}} + \left(1-\frac{e^{L(u_1)}}{1+e^{L(u_1)}}\right)\cdot\left(1-\frac{e^{L(u_2)}}{1+e^{L(u_2)}}\right)$$

$$= \frac{e^{L(u_1)}e^{L(u_2)}}{(1+e^{L(u_1)})(1+e^{L(u_2)})} + \left\{\frac{(1+e^{L(u_1)})-e^{L(u_1)}}{1+e^{L(u_1)}}\right\}\cdot\left\{\frac{(1+e^{L(u_2)})-e^{L(u_2)}}{1+e^{L(u_2)}}\right\}$$

$$= \frac{e^{L(u_1)}e^{L(u_2)}}{(1+e^{L(u_1)})(1+e^{L(u_2)})} + \left(\frac{1}{1+e^{L(u_1)}}\right)\cdot\left(\frac{1}{1+e^{L(u_2)}}\right)$$

$$P(u_1 \oplus u_2 = +1) = \frac{e^{L(u_1)}e^{L(u_2)}+1}{(1+e^{L(u_1)})(1+e^{L(u_2)})} \tag{4.14}$$

It is then relatively simple to determine the probability $P(u_1 \oplus u_2 = -1)$ as:

$$P(u_1 \oplus u_2 = -1) = 1 - P(u_1 \oplus u_2 = +1)$$

$$= 1 - \frac{1 + e^{L(u_1)} e^{L(u_2)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})}$$

$$= \frac{(1 + e^{L(u_1)})(1 + e^{L(u_2)}) - 1 - e^{L(u_1)} e^{L(u_2)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})}$$

$$= \frac{1 + e^{L(u_2)} + e^{L(u_1)} + e^{L(u_1)} e^{L(u_2)} - 1 - e^{L(u_1)} e^{L(u_2)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})}$$

$$P(u_1 \oplus u_2 = -1) = \frac{e^{L(u_2)} + e^{L(u_1)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})} \tag{4.15}$$

From equations (4.14) and (4.15):

$$L(u_1 \oplus u_2) = ln \frac{P(u_1 \oplus u_2 = +1)}{P(u_1 \oplus u_2 = -1)} \tag{4.16}$$

$$= ln \left[\frac{\frac{1 + e^{L(u_1)} e^{L(u_2)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})}}{\frac{e^{L(u_1)} + e^{L(u_2)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})}} \right]$$

$$= ln \frac{1 + e^{L(u_1)} e^{L(u_2)}}{e^{L(u_1)} + e^{L(u_2)}} \tag{4.17}$$

As Hagenauer [HAG96] indicates, this can be approximated as:

$$L(u_1 \oplus u_2) \approx sign(L(u_1)) sign(L(u_2)) min(|L(u_1)|, |L(u_2)|) \tag{4.18}$$

Two ‘soft’ values are added using the operator [+] giving:

$$L(u_1 \oplus u_2) = L(u_1)[+]L(u_2) \tag{4.19}$$

Where

$$L(u)[+]\infty = L(u)$$

$$L(u)[+]-\infty = -L(u)$$

$$L(u)+[0] = 0$$

Following on from equation (4.19), it can be shown that

$$\sum_{[+]h=1}^{H} L(u_h) = L\left(\sum_{\oplus h=1}^{H} u_h\right) \tag{4.20}$$

Where H is used to indicate that equation (4.19) can be extended to any number of soft values. Directly from equations (4.16) and (4.20):

$$\sum_{[+]h=1}^{H} L(u_h) = ln\frac{P\left(\sum_{\oplus h=1}^{H} u_h = +1\right)}{P\left(\sum_{\oplus h=1}^{H} u_h = -1\right)} \tag{4.21}$$

which becomes:

$$= ln\frac{\prod_{h=1}^{H}(e^{L(u_h)}+1)+\prod_{h=1}^{H}(e^{L(u_h)}-1)}{\prod_{h=1}^{H}(e^{L(u_h)}+1)-\prod_{h=1}^{H}(e^{L(u_h)}-1)} \tag{4.22}$$

Using the relation $\tanh\left(\frac{x}{2}\right) = \frac{e^x-1}{e^x+1}$, equation (4.22) can be simplified to:

$$\sum_{[+]h=1}^{H} L(u_h) = ln\frac{1+\prod_{h=1}^{H} tanh\left(\frac{L(u_h)}{2}\right)}{1-\prod_{h=1}^{H} tanh\left(\frac{L(u_h)}{2}\right)}$$

$$= 2\tanh^{-1}\left(\prod_{h=1}^{H}\tanh\left(\frac{L(u_h)}{2}\right)\right) \tag{4.23}$$

As $\tanh^{-1}(x) = \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)$.

This is time consuming to calculate. A simpler approximation of this, using equation (4.18), is:

$$\sum_{[+]h=1}^{H} L(u_h) = L\left(\sum_{\oplus h=1}^{H} u_h\right) \approx \left(\prod_{h=1}^{H} sign(L(u_h))\right) \min_{h=1,\ldots,H}\left\{\left|L(u_h)\right|\right\} \tag{4.24}$$

Showing that the reliability of the sum of soft values is determined mostly by the smallest term.

The information $\boldsymbol{u}$ is mapped to the encoded bits $\boldsymbol{x}$. These encoded bits are received by the decoder as $\boldsymbol{y}$. All with the time index k. From this the log-likelihood ratio for the system is:

$$L(x_k \mid y_k) = \ln\frac{P(x_k = +1 \mid y_k)}{P(x_k = -1 \mid y_k)} \tag{4.25}$$

From Bayes' theorem, this is equivalent to:

$$L(x_k \mid y_k) = \ln\left(\frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)}\frac{P(x_k = +1)}{P(x_k = -1)}\right)$$

$$= \ln\frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)} + \ln\frac{P(x_k = +1)}{P(x_k = -1)} \tag{4.26}$$

Assuming the channel to be flat fading with Gaussian noise, the Gaussian pdf, G(z) can be applied,

$$G(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(z-q)^2}{2\sigma^2}} \tag{4.27}$$

With q representing the mean and σ^2 representing the variance. Showing that:

$$ln\frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)} = ln\frac{e^{-\frac{E_b}{N_o}(y_k - a)^2}}{e^{-\frac{E_b}{N_o}(y_k + a)^2}}$$

$$ln\frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)} = 4\frac{E_b}{N_o}ay_k \tag{4.28}$$

With $\frac{E_b}{N_o}$ representing the signal to noise ratio per bit and a being the fading amplitude (a = 1 for a non-fading Gaussian channel).

From equation (4.7) and (4.26), the log-likelihood ratio of x_k dependent on y_k is:

$$L(x_k \mid y_k) = 4\frac{E_b}{N_o}ay_k + L(x_k) = L_c(y_k) + L(x_k) \tag{4.29}$$

Where $L_c = 4\frac{E_b}{N_o}a$ is the channel reliability. Therefore $L(x_k|y_k)$ is the weighted received value, $L_c y_k$, summed with the log-likelihood value of x_k, $L(x_k)$.

The sequence $\boldsymbol{y}$ is initially received over the channel. Each component decoder estimates the information sequence from one of the pair of encoded bit streams received over the channel. For the first iteration, there is no *a priori* $L(\boldsymbol{u})$ sequence. Therefore, the first iteration has $L(\boldsymbol{u})$ set to the all-zero sequence, and only processes $L_C\boldsymbol{y}$, the weighted received sequence. The output from each of the component decoders comes in two parts, $\boldsymbol{u}'$ and $L(\boldsymbol{u}')$. Where $\boldsymbol{u}'$ is the estimated information sequence and $L(\boldsymbol{u}')$ is the associated log-likelihood ratio sequence. The modified metric for the SOVA decoder is derived below.

Viterbi's algorithm searches for a state sequence ($\boldsymbol{S}^{(l)}$), or information sequence ($\boldsymbol{u}^{(l)}$), that gives the maximum *a posteriori* probability $P(\boldsymbol{S}^{(l)}|\boldsymbol{y})$. With binary trellises, l is either 1 or 2, denoting the survivor path and its competitor respectively. Bayes' theorem states that the *a posteriori* probability is expressed as:

$$P(\boldsymbol{S}^{(l)} \mid \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{S}^{(l)})\frac{P(\boldsymbol{S}^{(l)})}{P(\boldsymbol{y})} \tag{4.30}$$

The received sequence $\boldsymbol{y}$ can be deleted as it is fixed for metric computation and is not reliant on l. Therefore the maximisation becomes:

$$P(\boldsymbol{S}^{(l)}) = \max_l P(\boldsymbol{y} \mid \boldsymbol{S}^{(l)})P(\boldsymbol{S}^{(l)}) \tag{4.31}$$

$P(s_k)$, the probability of the state sequence terminating at time k, can be written as:

$$P(s_k) = P(s_{k-1})P(s_k) = P(s_{k-1})P(u_k) \tag{4.32}$$

With $P(s_k)$ representing the probability of the state at time *k*, and $P(u_k)$ being the probability of the bit also at time *k*. This can be expanded to:

$$\max_l P(\boldsymbol{y} \mid \boldsymbol{S}^{(l)})P(\boldsymbol{S}^{(l)}) = \max_l \left\{ \prod_{i=0}^{k} P(y_i \mid s_{i-1}^{(l)}, s_i^{(l)})P(s_k^{(l)}) \right\} \tag{4.33}$$

Where $(s_{i-1}^{(l)}, s_i^{(l)})$ is the state transition between time $i - 1$ and time i and y_i represents the associated received channel values for the state transition. Substituting equation (4.32), rearranging and changing the boundaries of the equation, this can be re-written as:

$$\max_l P(\boldsymbol{y} \mid \boldsymbol{S}^{(l)})P(\boldsymbol{S}^{(l)}) = \max_l \left\{ P(s_{k-1}^{(l)}) \prod_{i=0}^{k-1} P(y_i \mid s_{i-1}^{(l)}, s_i^{(l)})P(u_k^{(l)})P(y_k \mid s_{k-1}^{(l)}, s_k^{(l)}) \right\} \tag{4.34}$$

Where $P\left(s_k^{(l)}\right)$ is the probability of the state in path *l* at time *k*. Note that the last term is included because of the change of boundaries, and that

$$P(y_k \mid s_{k-1}^{(l)}, s_k^{(l)}) = \prod_{j=1}^{N} P(y_{k,j} \mid x_{k,j}^{(l)}) \tag{4.35}$$

Simply substituting this into equation (4.34) gives:

$$\max_l \left\{ P(s_{k-1}^{(l)}) \prod_{i=0}^{k-1} P(y_i \mid s_{i-1}^{(l)}, s_i^{(l)})P(u_k^{(l)}) \prod_{j=1}^{N} P(y_{k,j} \mid x_{k,j}^{(l)}) \right\} \tag{4.36}$$

Applying natural logarithm and multiplying by 2 to both sides of the equation and including two constants that are independent of *l* does not alter the maximisation and results in:

$$\max_l \left\{ M_k^{(l)} \right\} = \max_l \left\{ M_{k-1}^{(l)} + \left[2 \ln P(u_k^{(l)}) - C_u \right] + \sum_{j=1}^{N} \left[2 \ln P(y_{k,j} \mid x_{k,j}^{(l)}) - C_y \right] \right\} \tag{4.37}$$

Where M_k represents the *metric* (or *score*) at time *k*,

$$\frac{M_{k-1}^{(l)}}{2} = ln\left(P(s_{k-1}^{(l)}) \prod_{i=0}^{k-1} P(y_i \mid s_{i-1}^{(l)}, s_i^{(l)}) \right) \tag{4.38}$$

And

$$C_u = ln\, P(u_k = +1) + ln\, P(u_k = -1) \tag{4.39}$$

$$C_y = ln(P(y_{k,j} \mid x_{k,j} = +1)) + ln(P(y_{k,j} \mid x_{k,j} = -1)) \tag{4.40}$$

Substituting equations (4.39) and (4.40) into equation (4.38) gives the SOVA metric:

$$M_k^{(l)} = M_{k-1}^{(l)} + \sum_{j=1}^{N} x_{k,j}^{(l)}\, ln \frac{P(y_{k,j} \mid x_{k,j} = +1)}{P(y_{k,j} \mid x_{k,j} = -1)} + u_k^{(l)}\, ln \frac{P(u_k = +1)}{P(u_k = -1)} \tag{4.41}$$

Which reduces to

$$M_k^{(l)} = M_{k-1}^{(l)} + \sum_{j=1}^{N} x_{k,j}^{(l)} L_C y_{k,j} + u_k^{(l)} L(u_k) \tag{4.42}$$

Where $M_k^{(l)}$ is the survivor metric at state l at time k, $x_{k,j}^{(l)}$ corresponds to trellis transition bit or bits x_j from state l at time k, L_C is the channel reliability value, $y_{k,j}$ represents the received parity bit or bits accompanying the transition bits x, u_k is the trellis transition data bit, equivalent to $x_{k,1}$ for systematic codes and $L(u_k)$ is any *a priori* information.

For systematic codes, like those used for the turbo encoder, the metric can further be modified to

$$M_k^{(l)} = M_{k-1}^{(l)} + u_k^{(l)} L_C y_{k,1} + \sum_{j=2}^{N} x_{k,j}^{(l)} L_{Ck,j} y_{k,j} + u_k^{(l)} L(u_k) \tag{4.43}$$

Where, the systematic trellis transition bit and corresponding received, weighted bit have been extracted from the summation. The metric includes values from the previous metric ($M_{k-1}^{(l)}$), the channel reliability (L_c) and an *a priori* value ($L(u_k)$). The input to the decoder (the *a priori* information used and the systematic received information) must therefore be subtracted from this *a posteriori* information to provide the *a priori* information for the subsequent decoder.

4.5.3 Maximum *A Posteriori* (MAP) Decoding

The BCJR or MAP algorithm was originally presented coincidentally in both [McA72] and [BAH72] and later in [BAH74]. It is a modified version of this that is used by each of the component decoders of the MAP turbo decoder. This algorithm is used to perform the symbol-by-symbol MAP decoding.

Again, the turbo decoder's decision is whether u_k = +1 or –1.

It is presumed that $u_k = +1$ if $P(u_k = +1 | \boldsymbol{y}) > P(u_k = -1 | \boldsymbol{y})$

And $u_k = -1$ otherwise.

Put more simply, it can be said that the decision $\hat{u}_k$ is given by:

$$\hat{u}_k = sign[\, L(u_k)\,] \tag{4.44}$$

Where

$$L(u_k) \cong log\left(\frac{P(u_k = +1 | \boldsymbol{y})}{P(u_k = -1 | \boldsymbol{y})}\right) \tag{4.45}$$

Is the soft information related to the hard decision u_k.

Considering a simple communications system, where N data bits $\boldsymbol{u} = (u_1, u_2, ..., u_k, ..., u_N)$ are used by the RSC encoder to produce a state sequence $\boldsymbol{s}_0^k = (s_0, s_1, ..., s_k)$ of equal length (where s_0^k represents the state sequence from time 0 to k). The state transitions within the encoder's trellis are governed by the transitional probabilities

$$p_k(s, s') = P(s_k = s | s_{k-1} = s'); \quad 0 \le s, s' \le M_s - 1 \tag{4.46}$$

The encoder output codeword is represented thus:

$$\boldsymbol{v}_1^k = (\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k), \qquad \text{where } \boldsymbol{v}_k = (v_{k,0}, v_{k,1}, ..., v_{k,n-1})$$

For example, the encoder output $\boldsymbol{v}_k$ corresponds to state s_k.. In this thesis, all RSC encoders output two bits (rate 1/2), a systematic and a parity. For completeness, $\boldsymbol{v}_k$ is shown in the equation above with multiple parity bits to account for RSC encoders of greater rates (1/3 or 1/4 component codes for example). Thus, in this thesis, $\boldsymbol{v}_k$ comprises the systematic bit $v_{k,0}$ and the parity bit $v_{k,1}$.

The encoder output codeword is determined by the probabilities

$$q_t(x_k \mid s', s) = P(x_k \mid s_{k-1} = s', s_k = s); \qquad 0 \le s, s' \le M_s - 1 \tag{4.47}$$

The encoder output sequence is BPSK modulated to produce $\boldsymbol{x}_1^k$ and is transmitted over a Gaussian channel. The received sequence is $\boldsymbol{y}_1^k$. The Gaussian channel can be defined thus:

$$P(\boldsymbol{y}_1^N \mid \boldsymbol{x}_1^N) = \prod_{j=1}^{N} R(\boldsymbol{y}_j \mid \boldsymbol{x}_j) \tag{4.48}$$

Where

$$R(\boldsymbol{y}_j \mid \boldsymbol{x}_j) = \prod_{i=0}^{n-1} P(y_{j,i} \mid x_{j,i}) \tag{4.49}$$

And

$$P(y_{j,i} \mid x_{j,i} = -1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_{j,i}+1)^2}{2\sigma^2}} \tag{4.50}$$

Likewise,

$$P(y_{j,i} \mid x_{j,i} = +1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_{j,i}-1)^2}{2\sigma^2}} \tag{4.51}$$

Where σ^2 is noise variance.

Beginning with equation (4.45) and incorporating it into the code's trellis gives:

$$P(u_k = -1 \mid \boldsymbol{y}) = \sum_{S^-} P(s_{k-1} = s', s_k = s \mid \boldsymbol{y})$$

$$P(u_k = +1 \mid \boldsymbol{y}) = \sum_{S^+} P(s_{k-1} = s', s_k = s \mid \boldsymbol{y})$$

Which can be modified, using Bayes theorem, to:

$$P(u_k = -1 \mid \boldsymbol{y}) = \sum_{S^-} \frac{P(s_{k-1} = s', s_k = s, \boldsymbol{y})}{P(\boldsymbol{y})}$$

$$P(u_k = +1 | \mathbf{y}) = \sum_{S^+} \frac{P(s_{k-1} = s', s_k = s, \mathbf{y})}{P(\mathbf{y})}$$

Where $s_k \in S$ is the state of the encoder at time *k*, S^+ is the set of ordered pairs *(s',s)* corresponding to all state transitions $(s_{k-1} = s') \rightarrow (s_k = s)$ brought about by the data input $u_k = +1$, and S^- is defined in the same way, for $u_k = -1$. The received noisy codeword is expressed as $\mathbf{y} = y_1^N = (y_1, y_2, ..., y_N)$. Therefore, equation (4.49) can be re-written as:

$$L(u_k) = log\left(\frac{\sum_{S^+} P(s_{k-1} = s', s_k = s, y_1^N) / P(y_1^N)}{\sum_{S^-} P(s_{k-1} = s', s_k = s, y_1^N) / P(y_1^N)}\right) \tag{4.52}$$

Note that it is possible to cancel the $P(y_1^N)$ values in equation (4.52). Consequently the joint probability

$$P(s', s, y_1^N) = P(s_{k-1} = s', s_k = s, y_1^N) \quad for\ s', s = 0,1,...,M_s - 1 \tag{4.53}$$

is required. To compute this, the following probabilities must be defined:

$$\alpha_k(s) = P(s_k = s, y_1^N) \tag{4.54}$$

$$\beta_k(s) = P(y_{k+1}^N | s_k = s) \tag{4.55}$$

$$\gamma_k^i(s', s) = P(u_k = i, s_k = s, y_k | s_{k-1} = s') \quad ; i = 0,1 \tag{4.56}$$

Equation (4.53) can then be re-written in the following form:

$$P(s_{k-1} = s', s_k = s, y_1^N) = P(y_{k+1}^N, y_k, y_1^{k-1}, s_k = s, s_{k-1} = s')$$

Applying Bayes' theorem, this becomes:

$$= P(y_{k+1}^N | y_k, y_1^{k-1}, s_k = s, s_{k-1} = s').P(y_k, y_1^{k-1}, s_k = s, s_{k-1} = s')$$

$$= P(y_{k+1}^N | s_k = s).P(y_k, y_1^{k-1}, s_k = s, s_{k-1} = s')$$

Substituting for equation (4.55) gives:

$$= \beta_k(s).P(y_k, y_1^{k-1}, s_k = s, s_{k-1} = s')$$

$$= \beta_k(s).P(s_k = s, y_k \,|\, s_{k-1} = s', y_1^{k-1}).P(s_{k-1} = s', y_1^{k-1})$$

$$= \beta_k(s).P(s_{k-1} = s', y_1^{k-1}).P(s_k = s, y_k \,|\, s_{k-1} = s')$$

By substituting equation (4.56), this becomes:

$$= \beta_k(s). \sum_{i \in (0,1)} \gamma_k^i(s', s).\alpha_{k-1}(s')$$

$$P(s', s, y_1^N) = \alpha_{k-1}(s').\beta_k(s). \sum_{i \in (0,1)} \gamma_k^i(s', s) \tag{4.57}$$

Therefore equation (4.52) can be written:

$$L(u_k) = log\left(\frac{\sum_{s^+} \alpha_{k-1}(s').\beta_k(s).\gamma_k^1(s', s)}{\sum_{s^-} \alpha_{k-1}(s').\beta_k(s).\gamma_k^0(s', s)} \right) \tag{4.58}$$

Which implies that the log likelihood of each data bit u_k is dependent upon α at the previous time step, the current β value and γ for all bit value possibilities at the current time step *k*.

To derive α,

$$\alpha_k(s) = P(s_k = s, y_1^k)$$

$$= \sum_{s'=0}^{M_s - 1} P(s_{k-1} = s', s_k = s, y_1^k)$$

$$\alpha_k(s) = \sum_{s'=0}^{M_s - 1} P(s_{k-1} = s', s_k = s, y_1^{k-1}, y_k)$$

Applying Bayes' theorem,

$$\alpha_k(s) = \sum_{s'=0}^{M_s-1} P(s_k = s, y_k \left| s_{k-1} = s', y_1^{k-1} \right.).P(s_{k-1} = s', y_1^{k-1})$$

The second joint probability is equivalent to equation (4.54), resulting in:

$$= \sum_{s'=0}^{M_s-1} \alpha_{k-1}(s') P(s_k = s, y_k \left| s_{k-1} = s' \right.)$$

$$= \sum_{s'=0}^{M_s-1} \alpha_{k-1}(s'). \sum_{i \in (0,1)} P(s_k = s, u_k = i, y_k \left| s_{k-1} = s' \right.)$$

$$\alpha_k(s) = \sum_{s'=0}^{M_s-1} \alpha_{k-1}(s'). \sum_{i \in (0,1)} \gamma_k^i(s', s) \quad for\ k = 1,2,...,N \tag{4.59}$$

$$\alpha_0(0) = 1\ and\ \alpha_0(s) = 0\ for\ s \neq 0$$

In other words, the forward recursion begins at state 0 at time $k = 0$.

Deriving β,

$$\beta_k(s) = P(y_{k+1}^N \left| s_k = s \right.)$$

$$= \sum_{s'=0}^{M_s-1} P(s_{k+1} = s', y_{k+1}^N \left| s_k = s \right.)$$

Once again applying Bayes' theorem produces:

$$= \sum_{s'=0}^{M_s-1} \frac{P(s_{k+1} = s', y_{k+1}^N, s_k = s)}{P(s_k = s)}$$

$$= \sum_{s'=0}^{M_s-1} \frac{P(s_{k+1} = s', y_{k+1}, y_{k+2}^N, s_k = s)}{P(s_k = s)}$$

$$= \sum_{s'=0}^{M_s-1} \frac{P(y_{k+2}^N \left| y_{k+1}, s_{k+1} = s', s_k = s \right.).P(y_{k+1}, s_{k+1} = s', s_k = s)}{P(s_k = s)}$$

$$= \sum_{s'=0}^{M_s-1} \frac{P(y_{k+2}^{N} \mid s_{k+1}=s').P(y_{k+1}, s_{k+1}=s', s_k=s}{P(s_k=s)})$$

$$\beta_k(s) = \sum_{s'=0}^{M_s-1} \frac{\beta_{k+1}(s').P(s_{k+1}=s', y_{k+1} \mid s_k=s).P(s_k=s)}{P(s_k=s)}$$

Cancelling like terms leaves:

$$= \sum_{s'=0}^{M_s-1} \beta_{k+1}(s').P(s_{k+1}=s', y_{k+1} \mid s_k=s)$$

$$= \sum_{s'=0}^{M_s-1} \beta_{k+1}(s'). \sum_{i\in(0,1)} P(u_{k+1}=i, s_{k+1}=s', y_{k+1} \mid s_k=s)$$

$$\beta_k(s) = \sum_{s'=0}^{M_s-1} \beta_{k+1}(s'). \sum_{i\in(0,1)} \gamma_{k+1}^{i}(s,s') \quad for\ k = N-1,...,1,0 \tag{4.60}$$

$$\beta_N(0) = 1\ and\ \beta_N(s) = 0\ for\ s \neq 0$$

Therefore, the backward recursion begins at state 0 at time k = N.

Deriving γ,

$$\gamma_k^i(s',s) = P(u_k=i, s_k=s, y_k \mid s_{k-1}=s')$$

Applying Bayes' theorem:

$$= \frac{P(u_k=i, s_k=s, y_k, s_{k-1}=s')}{P(s_{k-1}=s')}$$

$$= \frac{P(y_k \mid u_k=i, s_k=s, s_{k-1}=s').P(u_k=i, s_k=s, s_{k-1}=s')}{P(s_{k-1}=s')}$$

$$= \frac{P(y_k \mid x_k).P(u_k=i, s_k=s, s_{k-1}=s')}{P(s_{k-1}=s')}$$

$$= \frac{P(y_k \mid x_k).P(u_k=i \mid s_k=s, s_{k-1}=s').P(s_k=s, s_{k-1}=s')}{P(s_{k-1}=s')}$$

$$\gamma_k^i(s',s) = \frac{P(y_k|x_k).P(u_k = i|s_k = s, s_{k-1} = s').P(s_k = s|s_{k-1} = s').P(s_{k-1} = s')}{P(s_{k-1} = s')}$$

Cancelling like terms produces:

$$\gamma_k^i(s',s) = P(y_k|x_k).P(u_k = i|s_k = s, s_{k-1} = s').P(s_k = s|s_{k-1} = s')$$

Which becomes:

$$\gamma_k^i(s',s) = P(y_k|x_k).P(x_k|s_k = s, s_{k-1} = s').P(s_k = s|s_{k-1} = s') \tag{4.61}$$

4.5.4 Max-Log-MAP

The amount of memory required, and the number of operations involving exponential values and multiplicative procedures, means that the complexity of the MAP algorithm becomes prohibitive when implementing a turbo decoder.

The system can be simplified by employing the logarithms of the probabilities defined in equations (4.54), (4.55) and (4.56) and thus transforming any multiplicative operations to summations. This gives the following:

$$\bar{\alpha}_k(s) = \log \alpha_k(s) \tag{4.62}$$

$$\bar{\beta}_k(s) = \log \beta_k(s) \tag{4.63}$$

$$\bar{\gamma}_k^i(s',s) = \log \gamma_k^i(s',s) \tag{4.64}$$

This means that, with reference to equation (4.59), $\bar{\alpha}_k(s)$ becomes:

$$\bar{\alpha}_k(s) = \log \sum_{s'=0}^{M_s-1} \sum_{i \in (0,1)} e^{\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k^i(s',s)} \tag{4.65}$$

$$\bar{\alpha}_0(0) = 0 \text{ and } \bar{\alpha}_0(s) = -\infty \text{ for } s \neq 0$$

Likewise,

$$\bar{\beta}_k(s) = log \sum_{s'=0}^{M_s-1} \sum_{i\in(0,1)} e^{\bar{\beta}_{k+1}(s')+\bar{\gamma}^i_{k+1}(s,s')} \tag{4.66}$$

$$\bar{\beta}_N(0) = 0 \text{ and } \bar{\beta}_N(s) = -\infty \text{ for } s \neq 0$$

Inserting these values into equation (4.58) yields:

$$L(u_k) = log \frac{\sum_{s=0}^{M_s-1} e^{\bar{\alpha}_{k-1}(s')+\bar{\gamma}^1_k(s',s)+\bar{\beta}_k(s)}}{\sum_{s=0}^{M_s-1} e^{\bar{\alpha}_{k-1}(s')+\bar{\gamma}^0_k(s',s)+\bar{\beta}_k(s)}} \tag{4.67}$$

Using the approximation

$$log(e^{\delta_1} + e^{\delta_2} + ... + e^{\delta_n}) \approx \max_{i\in\{1,2,..,n\}} \delta_i \tag{4.68}$$

Where $\max_{\{i\in1,2,...,n\}} \delta_i$ is found by successively computing *(n-1)* maximum function over only two values, Means that equation (4.67) can be estimated as:

$$L(u_k) \approx \max_s \left[\bar{\gamma}^1_k(s',s) + \bar{\alpha}_{k-1}(s') + \bar{\beta}_k(s)\right] - \max_s \left[\bar{\gamma}^0_k(s',s) + \bar{\alpha}_{k-1}(s') + \bar{\beta}_k(s)\right] \tag{4.69}$$

4.5.5 Log-MAP

The Max-Log-MAP algorithm approximates the log-likelihood ratio $L(u_k)$, and is therefore suboptimal. The Log-MAP algorithm instead uses the Jacobian algorithm [ERF94], to improve the log-likelihood ratio and therefore improve the decoding abilities of the algorithm with only a small increase in complexity:

$$\begin{aligned} log(e^{\delta_1} + e^{\delta_2}) &= max(\delta_1, \delta_2) + log(1 - e^{-|\delta_2 - \delta_1|}) \\ &= max(\delta_1, \delta_2) + f_c(|\delta_1 - \delta_2|) \end{aligned} \tag{4.70}$$

Where $f_c(.)$ is a correction function.

Using equation (4.70) recursively, an exact solution to the expression of equation (4.68) can be obtained:

$$\begin{aligned} log(e^{\delta_1}+...+e^{\delta_n}) &= log(\Delta+e^{\delta_n}),\ \Delta = e^{\delta_1}+...+e^{\delta_{n-1}} = e^{\delta} \\ &= max(log\,\Delta,\delta_n)+ f_c(|log\,\Delta-\delta_n|) \\ &= max(\delta,\delta_n)+ f_c(|\delta-\delta_n|) \end{aligned} \tag{4.71}$$

This then allows the calculation of $L(u_k)$ in equation (4.67), with

$$\delta_n = \bar{\gamma}_k^i(n',n)+\bar{\alpha}_{k-1}(n')+\bar{\beta}_k(n) \tag{4.72}$$

$n = 0,1,...,M_s-1, and\ i = 0,1$

By pre-calculating the correction function $f_c(.)$, and storing the resultant one-dimensional array ($f_c(.)$ is dependent on $|\delta-\delta_n|$ only), the complexity can be reduced.

4.5.6 Iterative MAP Decoding

Figure (4.9) describes an iterative decoding system for turbo codes. Beginning with the first component decoder in the system, when using the MAP decoding algorithm, the log-likelihood ratio for a rate *1/n* code is:

$$L_1(u_k) = log\frac{\sum_{s',s=0}^{M_s-1}\alpha_{k-1}(s')p_k^1(1)exp\left(-\frac{\sum_{j=0}^{n-1}(y_{k,j}-x_{k,j}^1(s))^2}{2\sigma^2}\right).\beta_k(s)}{\sum_{s',s=0}^{M_s-1}\alpha_{k-1}(s')p_k^1(0)exp\left(-\frac{\sum_{j=0}^{n-1}(y_{k,j}-x_{k,j}^0(s))^2}{2\sigma^2}\right).\beta_k(s)} \tag{4.73}$$

$p_k^1(1)$ and $p_k^1(0)$ represent the *a priori* information for 1 and 0 at the input to component decoder #1. In a similar way, $p_k^2(1)$ and $p_k^2(0)$ denote the *a priori* probabilities at the second component decoder. The MAP decoder assumes:

$$p_k^1(1) = p_k^1(0) = 1/2 \tag{4.74}$$

Equation (4.73) can also be written:

$$L_1(u_k) = \log \frac{p_k^1(1)}{p_k^1(0)} + \log \frac{\sum_{s',s=0}^{M_s-1} \alpha_{k-1}(s') \exp\left(-\frac{(y_{k,0} - x_{k,0}^1)^2 + \sum_{j=1}^{n-1} (y_{k,j} - x_{k,j}^1(s))^2}{2\sigma^2} \right) .\beta_k(s)}{\sum_{s',s=0}^{M_s-1} \alpha_{k-1}(s') \exp\left(-\frac{(y_{k,0} - x_{k,0}^0)^2 + \sum_{j=1}^{n-1} (y_{k,j} - x_{k,j}^0(s))^2}{2\sigma^2} \right) .\beta_k(s)} \tag{4.75}$$

Where *(n-1)* is the number of parity bits (this means that a rate 1/2 component code would do away with the summation altogether).

The code is systematic and therefore $x_{k,0}^i = 1, i = 0,1$ is independent of trellis and state. Thus $x_{k,0}^1 = 1$ and $x_{k,0}^0 = -1$. Equation (4.75) now becomes:

$$L_1(u_k) = \log \frac{p_k^1(1)}{p_k^1(0)} + \frac{2}{\sigma^2} y_{k,0} + L_{e1}(u_k) \tag{4.76}$$

Where:

$$L_{e1}(u_k) = \log \frac{\sum_{s's=0}^{M_s-1} \alpha_{k-1}(s') \exp\left(-\frac{\sum_{j=1}^{n-1} (y_{k,j} - x_{k,j}^1(s))^2}{2\sigma^2} \right) .\beta_k(s)}{\sum_{s's=0}^{M_s-1} \alpha_{k-1}(s') \exp\left(-\frac{\sum_{j=1}^{n-1} (y_{k,j} - x_{k,j}^0(s))^2}{2\sigma^2} \right) .\beta_k(s)} \tag{4.77}$$

And is the *extrinsic information*. As $L_{e1}(u_k)$ does not contain the received data input $y_{k,0}$, which is input to the second component decoder and would therefore cause correlation, it can be used as the *a priori* information for the next decoder.

The inputs to the second component decoder are represented here as $\tilde{y}_0$, the interleaved version of y_0, y_2, the corresponding parity from the second component encoder and $\tilde{L}_{e1}(u_k)$, the interleaved version of $L_{e1}(u_k)$.

$$\tilde{L}_{e1}(u_k) = log\frac{p_k^2(1)}{p_k^2(0)} \tag{4.78}$$

From this,

$$p_k^2(1) = 1 - p_k^2(0) \tag{4.79}$$

The *a priori* probabilities can be expressed as:

$$p_k^2(1) = \frac{e^{\tilde{L}_{e1}(u_k)}}{1 + e^{\tilde{L}_{e1}(u_k)}} \tag{4.80}$$

$$p_k^2(0) = \frac{1}{1 + e^{\tilde{L}_{e1}(u_k)}} \tag{4.81}$$

The second component decoder estimates $L_{e2}(u_k)$ and in a similar way to equation (4.76), the log-likelihood ratio for this decoder can be written:

$$L_2(u_k) = log\frac{p_k^2(1)}{p_k^2(0)} + \frac{2}{\sigma^2}\tilde{y}_{k,0} + L_{e2}(u_k) \tag{4.82}$$

Substituting equations (4.80) and (4.81) into (4.82), gives:

$$L_2(u_k) = L_{e1}(u_k) + \frac{2}{\sigma^2}\tilde{y}_{k,0} + L_{e2}(u_k) \tag{4.83}$$

$L_{e2}(u_k)$ is the extrinsic information from the second decoder, dependent upon the redundant information from the second component encoder. This can be used as the *a priori* information for the first decoder, at the beginning of the next iteration. Therefore equation (4.76) can be re-written as:

$$L_1(u_k) = \tilde{L}_{e2}(u_k) + \frac{2}{\sigma^2}\tilde{y}_{k,0} + L_{e1}(u_k) \tag{4.84}$$

4.6 Conclusions

This chapter outlines the evolution from basic information theory, through the concept of error control coding, to the methods and tools necessary for the design of turbo codes.

A brief discussion of the concept of error control coding, explaining modulo-q arithmetic and the theory of redundancy, was followed by the convolutional coding concept. Encoder variations and design were discussed followed by an explanation of the different methods of representation and their use in finding the error control properties of these codes. A description and worked example of the chief decoding method was also given.

Following this, the application of convolutional codes as component codes in the turbo encoding system was discussed, with special attention paid to the variations from conventional strategies when applying these codes to the turbo structure. This led to a description of the turbo encoder structure, giving the philosophy behind the inclusion of each component, their methods and design for the turbo code application.

Finally, the structure of the turbo decoder was discussed, the reasoning behind the design and a description of the iterative nature of the decoder. Finally, an introduction to the concept of Soft-Input Soft-Output (SISO) decoding was followed by descriptions and full derivations of the two competing decoding algorithms, SOVA and MAP, and their adaptation to turbo codes. The reasoning behind the further development of the MAP algorithm to Max-Log-MAP and Log-MAP was explained, with further derivations for each of these MAP derivatives.

Chapter

5

Turbo Codes Realisation And Software Implementation

5.1 Introduction

Chapter 4 presented a theoretical analysis and mathematical development of turbo codes, providing a basis for the design of a turbo coding system. This chapter investigates the implementation of the system and each of the different processes.

The chapter then discusses the considerations made during the design process, prior to the creation of the software. The completed systems are then compared with published results for both decoder strategies using long frames, providing a comparison of the new programs with known results. The effects of AWGN and Rayleigh fading channels on short frame turbo codes are then considered with an analysis of how increases in component code constraint length, number of iterations and changes in puncturing structure alter performance.

The penultimate section of this chapter investigates the complexity of each decoding algorithm, which leads to an evaluation of these decoders based on number of operations rather than number of iterations.

Finally, the effects of inaccuracies in Signal-to-Noise Ratio (SNR) estimation on the SOVA algorithm are considered as these have not been previously investigated for this algorithm.

5.2 Design Considerations

This software system is modular and as universal as possible, while remaining within the realms of conventional turbo code systems, to allow its use in future projects. To this end, the software produces results for any binary convolutional component code, punctured or not, with varying frame length in a standard two component code turbo system.

Modulation is Binary Phase-Shift Keying (BPSK). The design incorporated three different channels, standard Additive White Gaussian Noise (AWGN), Rayleigh fading with AWGN and a SUI 3-tap dispersive channel. The worked example found here uses AWGN for simplicity.

The two component decoders implemented were SOVA and Log-MAP.

The design of a turbo code system must consider many constraining aspects. To begin with, the type of channel distortion that will be encountered dictates whether other external components must be combined with the turbo decoder, as it requires accurate channel information to achieve its full potential.

The channel bandwidth is also an important consideration if turbo codes are to be further integrated into terrestrial mobile radio communications and for this reason puncturing should be considered. The importance of puncturing increases as more component codes are incorporated into the system or component codes with rates lower than those used here are utilised, this is due to the fact that a lower rate code requires more bandwidth.

The interleaver has a great effect on the performance of the turbo code. Ryan [RYA99] states that for large frame lengths (over 1000 bits), one pseudo random interleaver is as good as any other. This statement may stand for an infinite number of transmitted symbols, but research into the subject is still ongoing [BAR94], [SAI99] and improvements are still being found as not all channels are the same, nor all applications, and it stands to reason that certain interleavers will outperform others for a specific task.

For smaller frame lengths, such as those used in this project, the importance of the interleaver increases. For this reason, the method proposed by Pietrobon and Barbulescu [BAR94] as found in chapter 4.4.4 has been used here, although the process was not exhaustive and it is more than likely that there are better configurations still to be found. The interleavers used in this project were better than the average.

The academic bent has always been to search for higher performance and, with modern processing power, this is certainly possible. Nevertheless, the designer may want to consider this type of code against others on a complexity basis where a particular performance level at a particular signal-to-noise level is deemed satisfactory, as is the situation with voice. This being the case, the component code might also be a consideration, especially if the data to be encoded is not random, as, like the interleaver, the component encoder can be selected such that it produces code words with the higher code word weights than its competitors.

The inclusion of a user interface allowing specification of component code, frame length, puncturing, decoder type and number of iterations, channel properties (signal-to-noise ratio, channel

type) and simulation duration (number of frames transmitted) meant that a wide array of possible systems could be tested without recreating each system, both for this project and for those in the future. Woodard and Hanzo [WOO00], give an excellent description of the whole turbo code process.

5.3 Validation of Software

Results from the new software created for this project were validated by comparing with published results for long frame turbo codes. These can be seen below.

[HAY01] shows results for Log-MAP decoded turbo codes with data frames of 1200 bits, using $[7;5]_8$ component codes transmitted over AWGN using a block interleaver. Results are shown after 10 iterations.

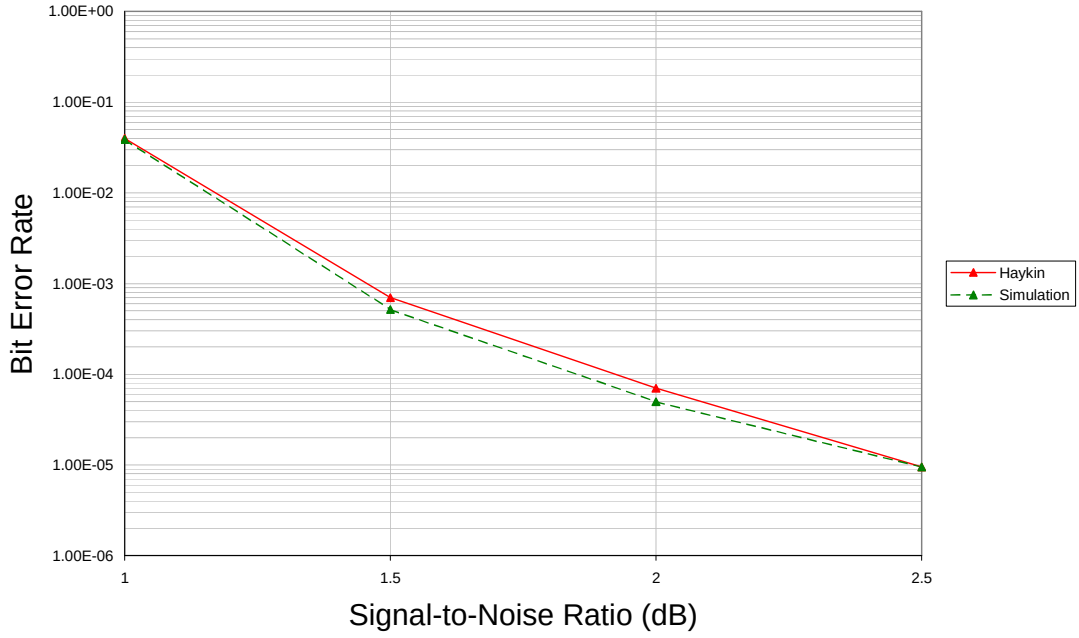


Figure 5.1a: Haykin Log-MAP Comparison

[VUC00] shows memory 2 turbo codes with frames of 1024 bits transmitted over an AWGN channel after 12 iterations using SOVA

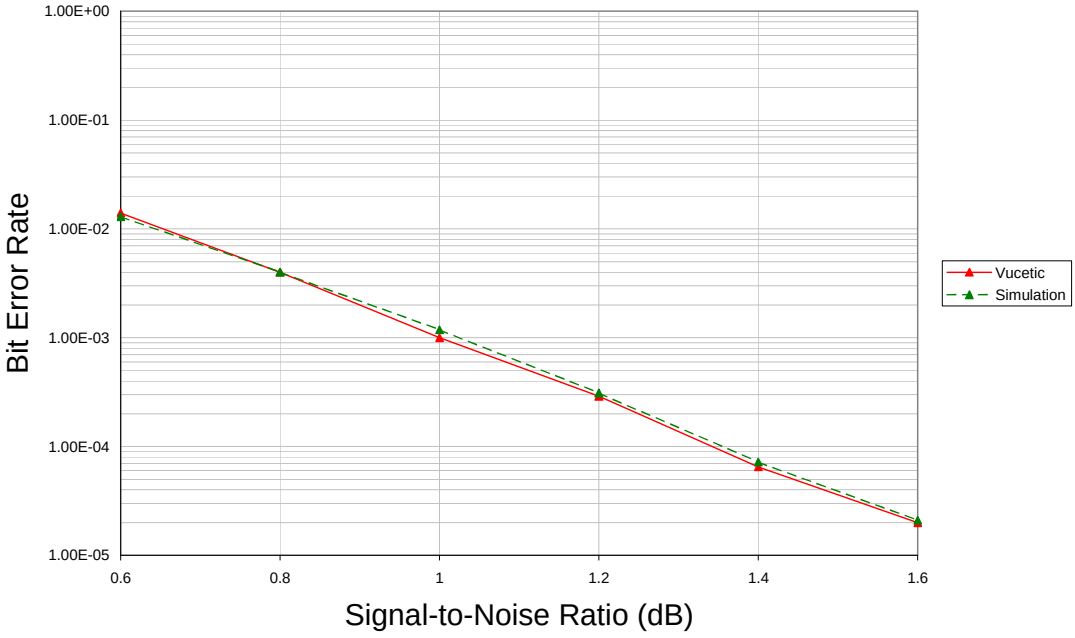


Figure 5.1b: Vucetic SOVA Comparison

Note that the results of the two comparisons in figures 5.1a and 5.1b do not match exactly. Part of the reason for this is that both sets of published results were manually recovered from graphs in printed material, introducing a certain amount of discrepancy. Considering figure 5.1a, the Log-MAP simulation in [HAY01] was run for only 20 frame errors per SNR, a relatively low level of accuracy compared to the results obtained through the software created here, where the number of frame errors per SNR varied between 43 and 62 (The new simulations were terminated on number of bit errors rather than frame errors). Considering the SOVA results of figure 5.1b, the interleaver used in the results obtained from [VUC00] was not specified. The results produced here for comparison were obtained using a pseudo-random interleaver. The authors of [VUC00] also did not specify the point at which the simulations were terminated. Under these circumstances, it is reasonable to say that the software developed for this project is producing results comparable to those in the public domain.

5.4 Simulations

Primary simulations were made over Added White Gaussian Noise (AWGN) channels. All simulations used Binary Phase Shift Keying (BPSK) modulation. Where puncturing was applied, the standard system as described in 4.4.5 was used.

For each interleaver length, a reasonable number of iterations of the interleaver selection process as defined in chapter 4.4.4 were run in order to find a suitable interleaver. The chosen interleavers have been utilised throughout this project. No claim is made here that they are optimal as an exhaustive search was not made, rather they are referred to as better than average.

The decision was made to use eight iterations because after this, performance increases begin to reduce. Figures 5.2a and 5.2b below help to highlight this and it can be seen that the Signal-to-Noise ratio gain from four iterations to eight iterations at a Bit Error Rate of 10^{-4} is approximately equal to the gain from three iterations to four, Likewise at a Frame Error Rate of 10^{-2}.

Four different codes were considered. The commonly used 4-state $[7;5]_8$ code, and the $[15;17]_8$ 8-state code were selected from [BEN96a] and have the largest effective free distances for codes with their constraint lengths and rates ($d_{free,eff}$ = 10 and 14 respectively). The $[23;33]_8$ 16-state code has not been found in the published materials, but does have the maximum possible effective free distance for a code with its parameters ($d_{free,eff}$ = 22). The fourth code has generator $[63;75]_8$ and 32-states. This code is not an optimal code for turbo codes with an effective free distance of 30 where the optimal is 38. In choosing this component, it was possible to examine how a longer code with a higher, but sub-optimal $d_{free,eff}$, would perform against shorter codes with smaller, but optimal $d_{free,eff}$. Each simulation was conducted for 200 bit errors per signal-to-noise ratio and the two decoding algorithms compared were SOVA and Log-MAP as described in the previous chapter and explained earlier in this chapter.

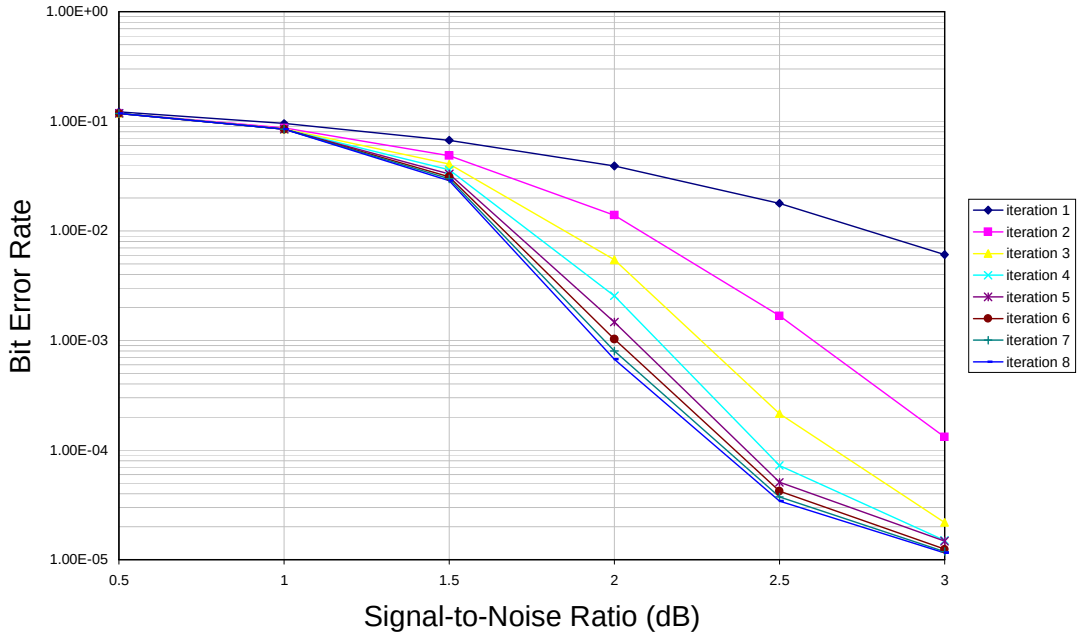


Figure 5.2a: Example Bit Error Rates for $[7;5]_8$ code after 8 Turbo Decoder Iterations

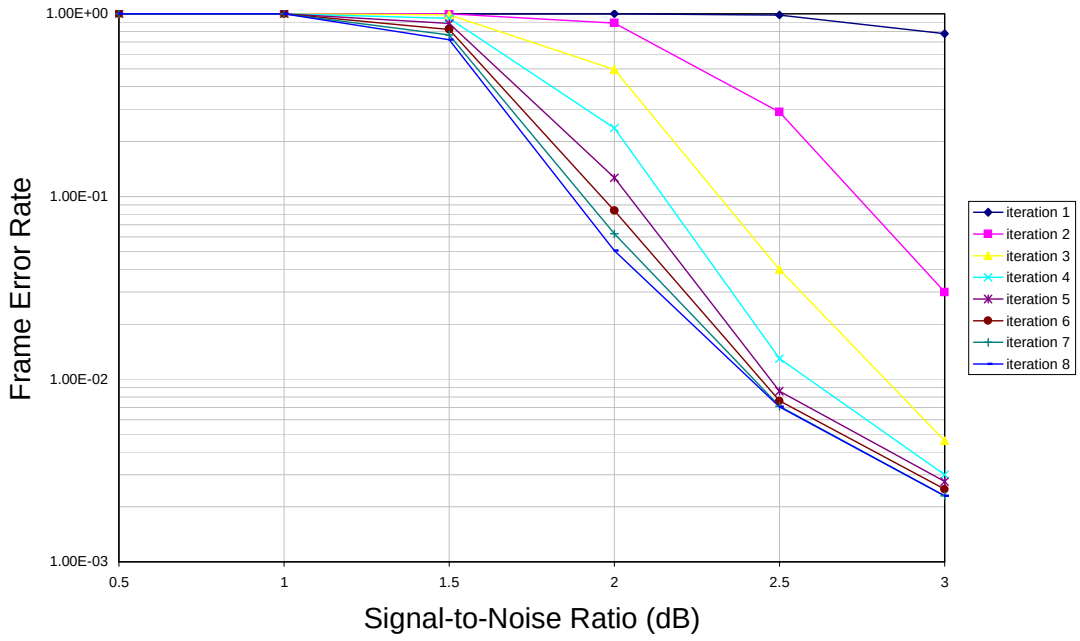


Figure 5.b: Example Frame Error Rates for $[7;5]_8$ code after 8 Turbo Decoder Iterations

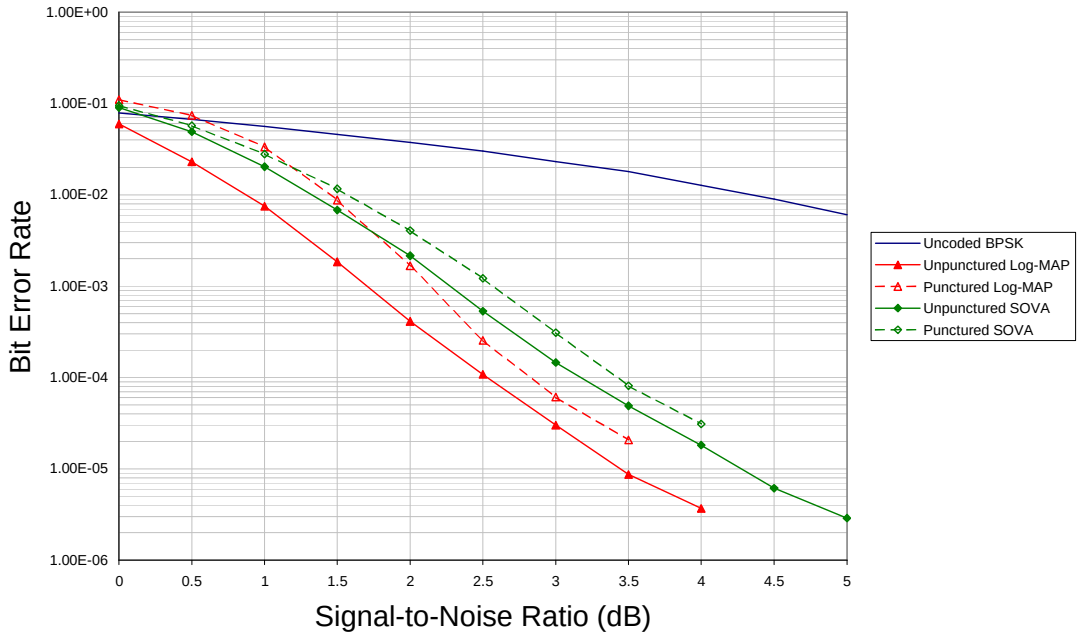


Figure 5.3: SOVA vs. Log-MAP Bit Error Rate for $[7;5]_8$ Component Code over AWGN for 100 Bit Data Frames after 8 Decoder Iterations

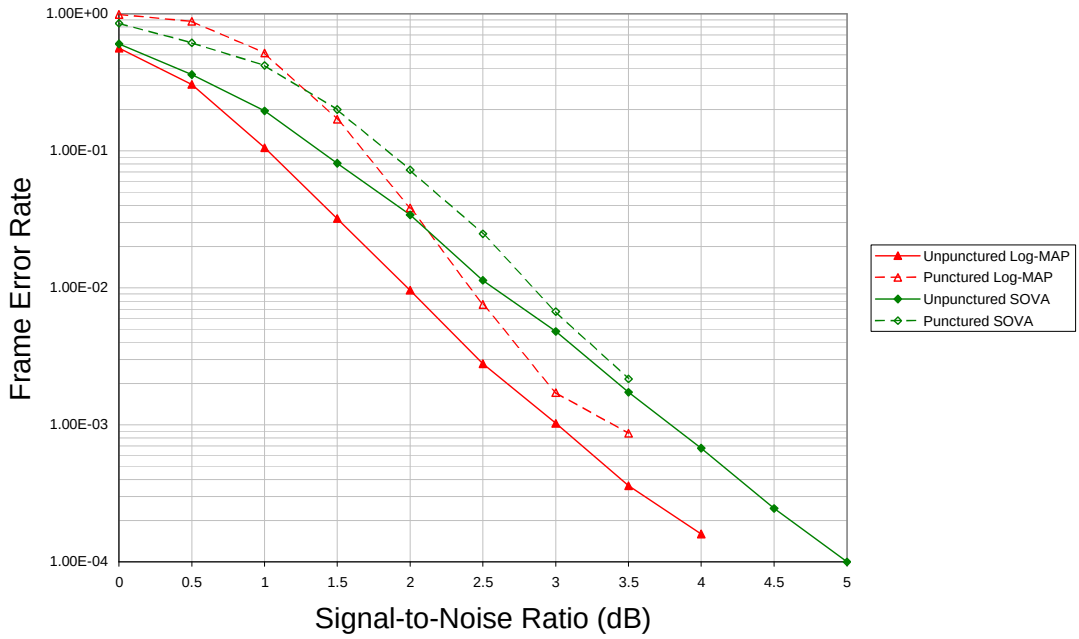


Figure 5.4: SOVA vs. Log-MAP Frame Error Rate for $[7;5]_8$ Component Code over AWGN for 100 Bit Data Frames After 8 Decoder Iterations

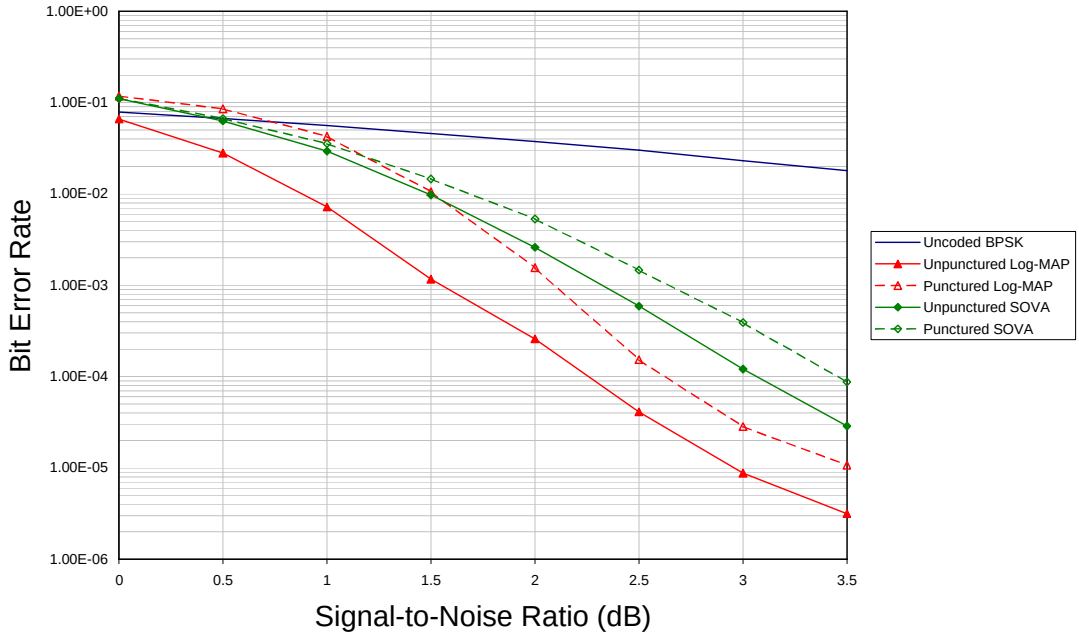


Figure 5.5: SOVA vs. Log-MAP Bit Error Rate for $[15;17]_8$ Component Code over AWGN for 100 Bit Data Frames after 8 Decoder Iterations

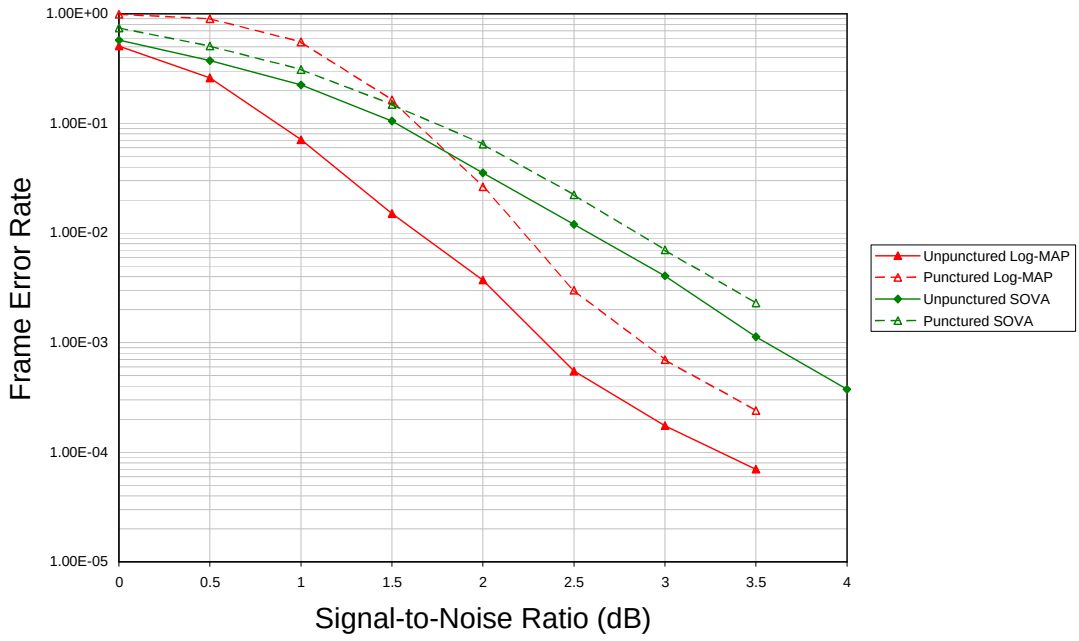


Figure 5.6: SOVA vs. Log-MAP Frame Error Rate for $[15;17]_8$ Component Code over AWGN for 100 Bit Data Frames After 8 Decoder Iterations

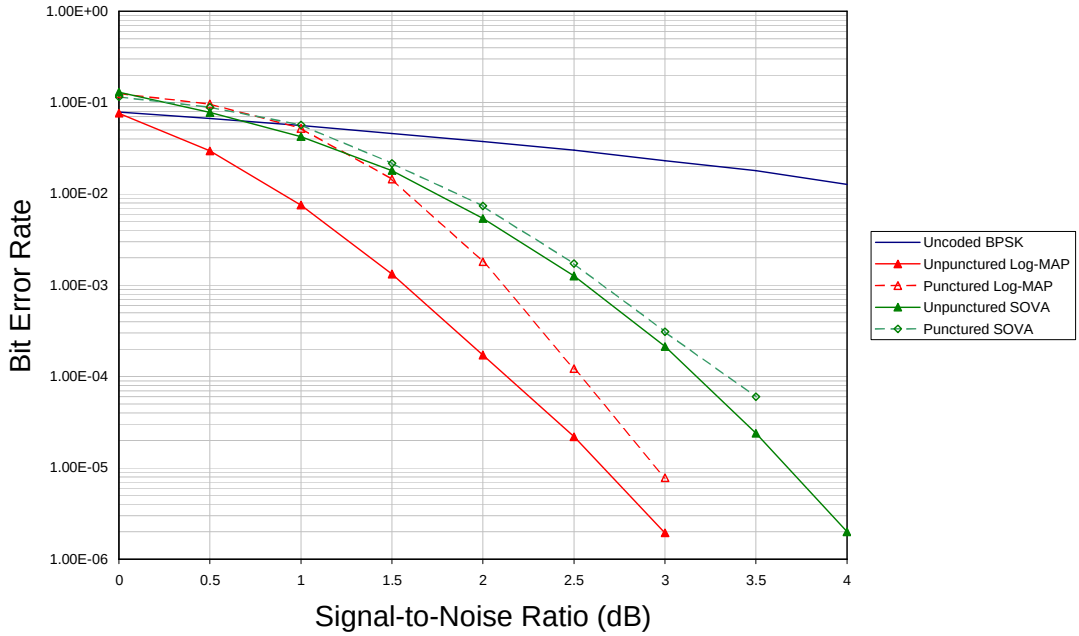


Figure 5.7: SOVA vs. Log-MAP Bit Error Rate for $[23;33]_8$ Component Code over AWGN for 100 Bit Data Frames after 8 Decoder Iterations

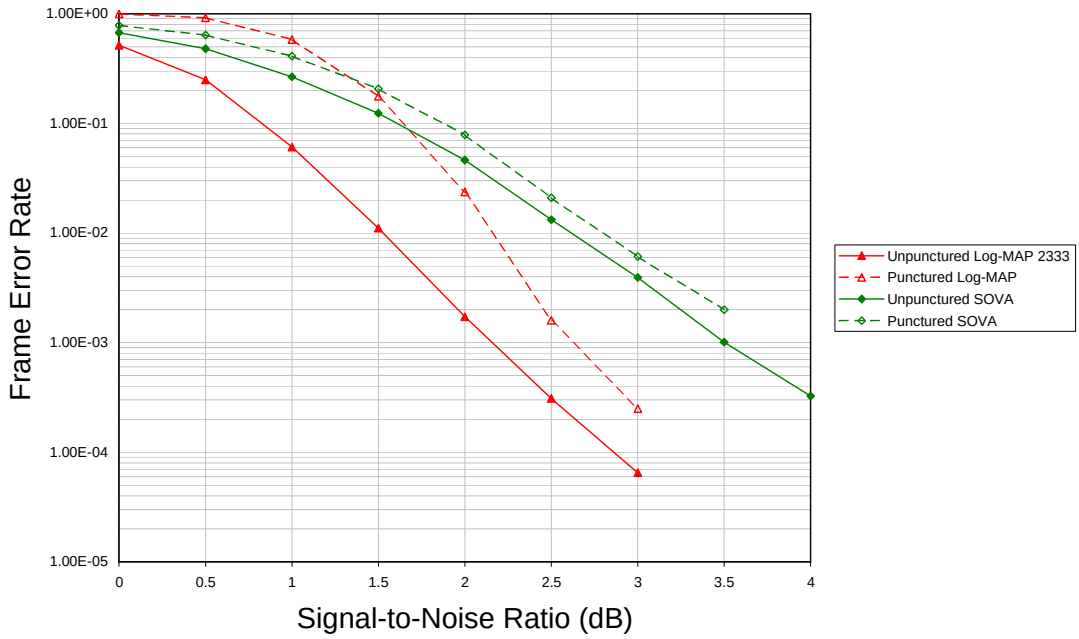


Figure 5.8: SOVA vs. Log-MAP Frame Error Rate for $[23;33]_8$ Component Code over AWGN for 100 Bit Data Frames After 8 Decoder Iterations

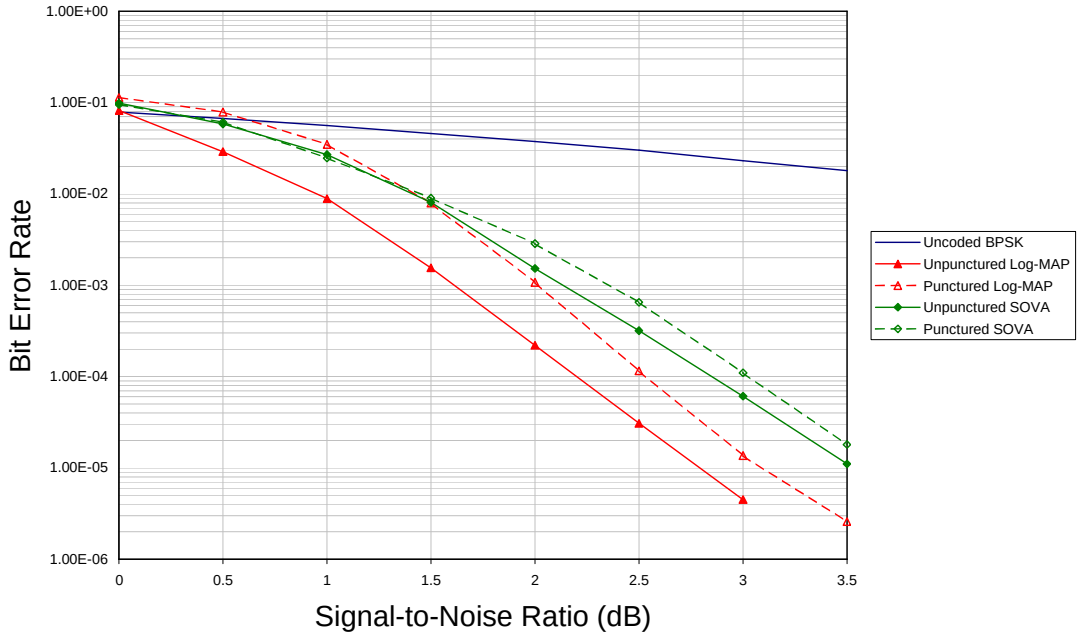


Figure 5.9: SOVA vs. Log-MAP Bit Error Rate for $[63;75]_8$ Component Code over AWGN for 100 Bit Data Frames after 8 Decoder Iterations

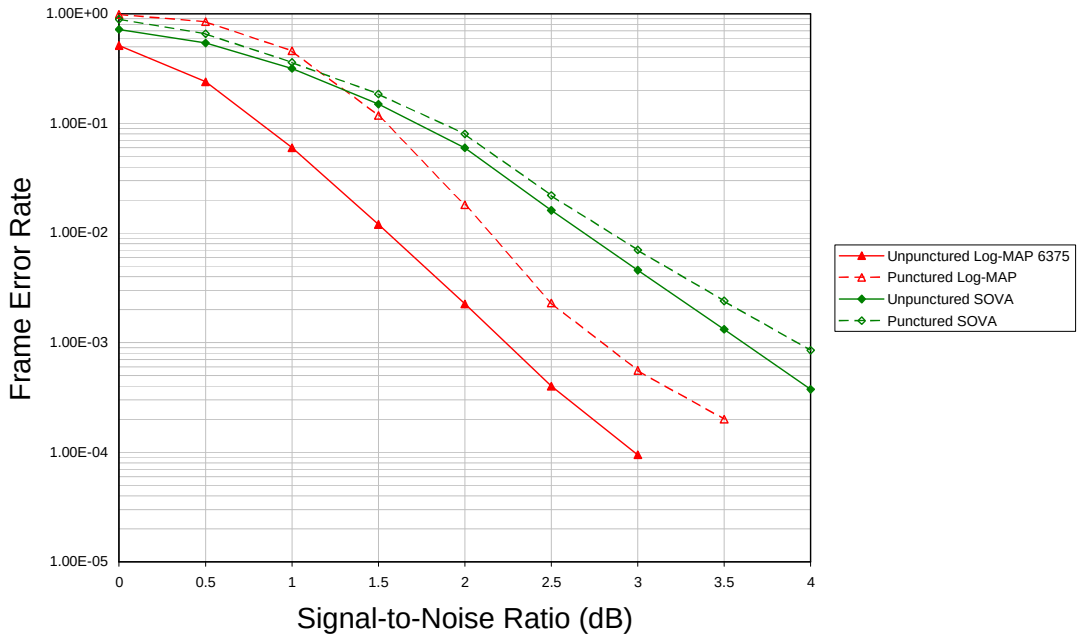


Figure 5.10: SOVA vs. Log-MAP Frame Error Rate for $[63;75]_8$ Component Code over AWGN for 100 Bit Data Frames After 8 Decoder Iterations

5.4.1 Analysis of Turbo Codes with 100 Bit Datawords over AWGN

Table 5.1 shows the points at which each of the four turbo codes attains Bit Error Rates sufficient for voice (10^{-2}) and data (10^{-4}) transmissions with the shorter, 100 bit, data frames for simulations with and without puncturing.

Code	Log-MAP 10^{-2}	SOVA 10^{-2}	Log-MAP 10^{-4}	SOVA 10^{-4}
$[7;5]_8$ Un-punctured	0.85dB	1.3dB	2.467dB	3.1dB
$[7;5]_8$ Punctured	1.433dB	1.567dB	2.7dB	3.34dB
$[15;17]_8$ Un-punctured	0.866dB	1.476dB	2.25dB	3.06dB
$[15;17]_8$ Punctured	1.512dB	1.7dB	2.619dB	3.464dB
$[23;33]_8$ Un-punctured	0.878dB	1.73dB	2.108dB	3.162dB
$[23;33]_8$ Punctured	1.581dB	1.838dB	2.541dB	3.338dB
$[63;75]_8$ Un-punctured	0.952dB	1.381dB	2.226dB	2.881dB
$[63;75]_8$ Punctured	1.405dB	1.429dB	2.53dB	3.06dB

Table 5.1: Bit Error Rate Observations for 100 Bit Data Frame Simulations

From the figures above, it is clear that the un-punctured turbo codes outperform their punctured counterparts, albeit at a severe loss in code rate. As the punctured codes have only one parity bit per symbol while the un-punctured codes have two, the un-punctured codes can be said to have 1.5 times the information of the punctured version. The puncturing mechanism used here did not exhibit 'even parity bit protection' as suggested in [HO98b] and therefore the gains made could be reduced fairly simply, by altering the interleaver design. The error floors exhibited, most obviously in the punctured Log-MAP decoder results (although the bit error rates at which these occurred were not obtained by the SOVA decoder), are symptomatic of badly designed interleavers and would suggest that the interleaver selection process used here is either flawed for such small frame lengths or was simply not run for a sufficient number of searches. The fact that the effects are more obvious with punctured codes suggests that efficient interleavers for un-punctured turbo codes are not necessarily efficient for punctured versions of the same codes. The use of puncturing is not taken into account by the method used in this body of work when designing the interleaver, which suggests that the method is indeed flawed and requires revision.

The first point to notice is that the bit error rate gains brought about by the reduction in code rate are smaller for the SOVA decoder. Un-punctured Log-MAP has gains at 10^{-2} of 0.583dB, 0.646dB, 0.703dB and 0.453dB as the component code constraint lengths increase, whereas SOVA has gains of 0.267dB, 0.224dB, 0.108dB and 0.048dB for the same criteria. This continues, to certain extent, at the lower point of interest with Log-MAP showing gains of 0.233dB, 0.369dB, 0.433dB and 0.304dB for the same codes while SOVA exhibits gains of 0.24dB, 0.404dB, 0.176dB and 0.179dB. It can be seen from the latter results that the Log-MAP gains are not always larger than the SOVA gains, however, in the two instances where the SOVA gain is larger (4-state and 8-state code), the increase is slight.

These gains could suggest that SOVA copes better with the loss of information that puncturing brings, on the other hand, they may suggest that SOVA does not make full use of the extra information supplied when the received information is not punctured. The first possibility is the most likely, as it would be expected that a decoder that does not make full use of available information would perform at a much-reduced level when less information was available to begin with. This is also an indication of the methods that the two decoders use. Although both the Viterbi based and MAP based algorithms work to find the maximum likelihood path, they do it in different ways. Viterbi decoders work to minimise the word error probability whereas MAP decoders minimise the bit (or symbol) error probability [VUC00]. With less information available due to puncturing, errors encountered at a particular point in time by the MAP decoder are not necessarily part of the final sequence output by the Viterbi decoder.

Considering the gains obtained by the Log-MAP algorithm over SOVA at the lower point of interest as percentage increases of the gains made at the higher point of interest, it is evident that the rate at which these decoder gains grow, with signal-to-noise ratio, reduces as optimal effective free distance increases. Where puncturing was omitted, the $[7;5]_8$ Log-MAP code produced a gain over the equivalent SOVA at 10^{-4} that is 141% that which was obtained at 10^{-2}, whereas the $[15;17]_8$ code produced a gain increase of 133% and the $[23;33]_8$ code produced a gain increase of 124% under

similar circumstances. The simulations that included puncturing acted in the same way, although the percentage increases were much greater, with increases in gains for the same codes being 478%, 450% and 310%. The 32-state code did not follow this order, producing a percentage increase of 158% where no puncturing was used and 2208% where puncturing was used. This large percentage increase was due to the fact that the two decoding algorithms performed in a similar way to one another at the higher point of interest, producing a very small gain for Log-MAP over is counterpart. This small advantage for Log-MAP increased at the lower point of interest. Therefore, even though the gains that this code created for Log-MAP in comparison to SOVA at the lower BER were both amongst the smallest gains of the four codes at this point of interest and very similar in magnitude (with the un-punctured simulation returning a gain of 0.655dB at 10^{-4} and the punctured simulation returning a gain of 0.53dB at the same point), they produced the largest percentage increase. This suggests that codes that are not wisely chosen with respect to $d_{free,eff}$ will not only perform badly compared to codes with smaller, but optimal, $d_{free,eff}$ but will also exhibit much smaller performance increases as the signal-to-noise ratio increases.

Considering the un-punctured results alone, one of the most obvious aspects is that Log-MAP consistently outperforms SOVA at all points on the curve. This follows results for longer frame codes in the literature. In addition, as expected, the Log-MAP decoder gains rise with constraint length as far as the codes with maximum $d_{free,eff}$ are concerned. Here the 4-state Log-MAP decoder displays a gain of 0.45dB over its SOVA counterpart, the 8-state shows a gain of 0.61dB and the 16-state produces a gain of 0.852dB at a bit error rate of 10^{-2}. These gains further increase at the lower BER of 10^{-4} with the same codes giving 0.633dB, 0.81dB and 1.054dB at this point. However, the 32-state code, with an effective free distance far below the maximum for a code with its constraint length, produces the lowest decoder gain at the voice quality BER with 0.429dB and very nearly the lowest decoder gain at the data quality BER with 0.655dB, even though it has the greatest effective free distance of the codes examined. This serves to illustrate one of the reasons why it is more important to choose a code with optimal $d_{free,eff}$ for its constraint length over a code with a greater effective free distance that is not optimised to its constraint length.

It is also interesting to note that, for both decoders, neither the code with the greatest effective free distance overall nor the code with greatest optimal effective free distance produces the best performance at the higher, voice quality, BER. In fact, the order of performance, for both decoders, at this bit error rate begins with the 4-state code, followed by the 8-state and 16-state codes and finishing with the 32-state code where Log-MAP decoding was used. The SOVA decoding results were slightly different. The 4-state code was followed by the 32-state code, the 8-state code and finally the 16-state code. This suggests that short-frame turbo codes with small, optimal, effective free distances perform better at voice quality BERs than both those codes with larger sub-optimal effective free distances and those with larger optimal effective free distances.

It seems that the decoders differ more obviously at the lower bit error rate. Although the Log-MAP decoder still outperforms the SOVA decoder, the order in which performance improves with respect to the component codes used differs. The SOVA decoder order of performance begins with the

32-state code, followed by the 8-, 4- and 16-state codes. The Log-MAP decoder, on the other hand, attains the best performance with the 16-state code, followed by the 32-state, 8-state and 4-state codes. This being the case, the Log-MAP order of performance would be closer to that expected for conventional codes, those with higher numbers of states and larger effective free distances producing the best results. However, the observations in table 5.1 have been taken from figures 5.3-5.10 manually and the ranges from the best performing and worst performing codes for each decoder at this BER are just 0.359dB for the Log-MAP algorithm and 0.281dB for the SOVA algorithm. Allowing for inaccuracies in measurement, the validity of the previous statements may therefore be compromised.

When considering the punctured results, the first observation is that the performance improvements obtained by the Log-MAP decoder over the SOVA decoder are drastically reduced compared to those made with un-punctured codes. The gains at voice quality BERs are 0.134dB for the $[7;5]_8$ code, 0.188dB for the $[15;17]_8$ code and 0.257dB for the $[23;33]_8$ code. The 32-state $[63;75]_8$ code shows a small gain of 0.024dB at this BER. At the lower BER, the gains are larger with 0.64dB for the 4-state code, 0.845dB for the 8-state code, 0.797dB for the 16-state code and 0.53dB for the 32-state code. It should be noted that although these values are much smaller than those obtained without puncturing, the percentage increases in decoder gain at the lower BER, when compared with the voice quality BER, are much larger than those found with the un-punctured codes, as discussed earlier. As with the un-punctured results, the rate of increase in decoder gain reduces with the increase in optimised effective free distance with the $[7;5]_8$ Log-MAP having a gain at 10^{-4} 478% that at 10^{-2}, the $[15;17]_8$ code having an equivalent increase of 450% and the $[23;33]_8$ code an increase of 310%.

Examining the results more closely, it is also apparent that the SOVA codes outperform the Log-MAP codes at very low signal-to-noise ratios, only beaten once the signal-to-noise ratio gets above around 1dB-1.5dB.

When puncturing is utilised, the two decoder strategies produce similar responses in terms of which component codes produce the best performance. The most apparent difference, when comparing these codes with their un-punctured counterparts in this way, occurs at the higher bit error rate. Here the order of performance, with the code requiring the least signal-to-noise ratio first, begins with the 32-state code and is followed by the 4-state, 8-state and 16-state code. This is the same for both decoders. As with the un-punctured results, the codes with smaller optimal effective free distances outperform those with larger optimal effective free distances. What is different is that the 32-state code beats them all. As this is the code that has the highest number of states and constraint length, it would be expected to produce the best performance in conventional coding. However, examining the rest of the decoder performance sequence shows that the other codes follow the order of smallest optimal effective free distance to largest optimal effective free distance.

At the lower BER, the two decoders act in a similar way to each other, and in a similar way to conventional codes. Both decoders obtain the best performance with the 32-state code. The 16-state code is the second best performer. These codes are then followed by the 8-state and 4-state for Log-MAP decoders, with the positions of these two codes swapped when SOVA decoding was used. Under these circumstances, higher signal-to-noise ratios with puncturing, it can be seen that both decoders get

the best results with codes with larger $d_{free,eff}$, regardless of whether they are optimal for their constraint length or not. It would be expected, however, that a 32-state code with optimal effective free distance would outperform the 16-state code used here by a greater margin.

Table 5.2 below shows the signal-to-noise ratios at which FERs of 10^{-1} and 2×10^{-3} were obtained.

Code	Log-MAP 10^{-1}	SOVA 10^{-1}	Log-MAP 2×10^{-3}	SOVA 2×10^{-3}
$[7;5]_8$ Un-punctured	1.017dB	1.379dB	2.707dB	2.983dB
$[7;5]_8$ Punctured	1.707dB	1.862dB	2.983dB	3.621dB
$[15;17]_8$ Un-punctured	0.851dB	1.5dB	2.149dB	3.243dB
$[15;17]_8$ Punctured	1.622dB	1.703dB	2.608dB	3.514dB
$[23;33]_8$ Un-punctured	0.786dB	1.571dB	1.929dB	3.2dB
$[23;33]_8$ Punctured	1.629dB	1.843dB	2.429dB	3.457dB
$[63;75]_8$ Un-punctured	0.8dB	1.686dB	2dB	3.286dB
$[63;75]_8$ Punctured	1.529dB	1.829dB	2.514dB	3.514dB

Table 5.2:Frame Error Rate Observations for 100 Bit Data Frame Simulations

As with the bit error results, the un-punctured codes outperform the punctured, with the un-punctured Log-MAP codes producing the best frame error rate results overall.

The frame error rate curves generally confirm the conclusions drawn from the BER results. The gains obtained by omitting the puncturing mechanism remain smaller for SOVA in much the same way as was evident in the bit error rate curves. The gains made by omitting the puncturing mechanism in the Log-MAP decoder at 10^{-1} were 0.362dB for the 4-state code, 0.771dBfor the 8-state code, 0.833dB for the 16-state code and 0.739 for the 32-state code. The equivalent gains in the SOVA decoder were 0.483dB, 0.203dB, 0.272dB and 0.143dB respectively. The situation remained unchanged at 2×10^{-3}, where the Log-MAP decoder without a puncturing mechanism gained 0.276dB for the 4-state code, 0.459dB for the 8-state code, 0.5dB for the 16-state code and 0.514dB for the 32 state code, over the same decoder using puncturing. The equivalent gains for the SOVA decoder were 0.155dB, 0.271dB, 0.257dB and 0.228dB respectively. Here the same effect is seen as in the bit error rate performance results, SOVA is less affected by puncturing than Log-MAP.

Comparing the decoder gains obtained with the Log-MAP algorithm in comparison with the SOVA algorithm at the two points of interest, it can be seen that the un-punctured codeword gains increase with rising constraint length regardless of whether the simulated code has optimal $d_{free,eff}$ or not. For un-punctured codes, at the higher point of interest, Log-MAP shows gains of 0.362dB, 0.649dB, 0.775dB and 0.896dB as the complexity of the component codes increases. At the lower point of interest, 2×10^{-3}, these gains increase to 0.759dB, 1.094dB, 1.271dB and 1.286dB for the same codes respectively. In the same way as they did in the bit error rate results, the rate of gain increases,

from 10^{-1} to 2×10^{-3}, reduce with an increase in effective free distance, optimal or otherwise. The un-punctured decoder gains at the lower frame error rate are 210% those obtained at the higher FER for the 4-state code, 169% for the 8-state code, 164% for the 16-state code and 143% for the 32-state code. This follows the BER results.

On the other hand, the frame error rate results for the punctured codes do not react in the same way as the bit error rate results. In order of rising constraint length, the Log-MAP decoder gains for these codes at a frame error rate of 10^{-1} are 0.155dB, 0.081dB, 0.214dB and 0.3dB. At the lower point of interest, 2×10^{-3}, the Log-MAP gains are 0.638dB, 0.906dB, 1.028dB and 1dB for the same codes respectively. This random pattern with respect to the order in which the decoder gains increase is borne out in the rates at which the gains increase. The percentage decoder gains in the FER curves at the lower point of interest with respect to the higher are 412%, 1119%, 480% and 333% as constraint length increases. So, in the case of puncturing, the changes in decoder gain produced by using the Log-MAP decoding algorithm instead of the SOVA cannot be tied to the constraint length or the effective free distance of the component codes, optimal or otherwise.

When examining the orders of performance at these points of interest and comparing them to those found when considering the bit error rate curves, some differences are also found. To begin with, looking at the un-punctured Log-MAP order of performance, at the higher FER, the order of performance is the reverse of that obtained at the higher BER. At the lower frame error rate, the order of performance is unchanged from that obtained in the BER results. In the case of the un-punctured SOVA, the order of performance at the higher FER remains largely unchanged except for the fact that the 32-state code produces the worst results.

Generally, the punctured code orders of performance are more similar to those found when examining the BER performance curves. The highest performing codes at both points of interest in the frame error rate results are the same as those in the bit error rate results, with the exception of the SOVA decoder at the lower frame error rate. The best performer in terms of frame error rate at this point and for this decoder is the 16-state code, whereas this actually performed worse than the 32-state at a similar signal-to-noise ratio when bit error rate was considered.

5.4.2 Conclusions for Turbo Codes with 100 Bit Datawords over AWGN

In conclusion, for 100 bit data frame turbo codes, it appears that, to design a good turbo code, the application must be carefully considered. If the bandwidth is available, un-punctured turbo codes perform much better than those that use puncturing to reduce the coding rate. This is no surprise as the decoder has all possible information to work with. Where either bit error rate or frame error rate are the primary concern, the un-punctured Log-MAP decoder produces better results at any signal-to-noise ratio. The performance difference between un-punctured Log-MAP and SOVA decoders increases with the optimal effective free distance of the code. This difference also grows with signal-to-noise ratio.

The rate at which this difference increases will be smaller for a code with large optimal effective free distance than for a code with smaller optimal effective free distance.

The importance of choosing a code with optimal effective free distance for un-punctured turbo codes cannot be emphasised enough. In these simulations, the code with the largest effective free distance was non-optimal and its position in the order of performance, compared with codes with shorter constraint length was somewhat random. Where no puncturing was used, this code did not return favourable performance at voice quality bit error rates or at data quality bit error rates with Log-MAP decoding. This code did however, consistently produce the best results where puncturing was included in the simulations.

If voice quality bit error rates are required, with un-punctured turbo codes using small datawords, then it is better to choose a code with small optimal effective free distance over one with large effective free distance. In these results, the 4-state code outperformed all other optimised codes, followed by the 8-state code and 16-state code.

If data quality bit error rates are required, then, generally, codes with larger optimal effective free distances will produce better results.

If the bandwidth is more constricted and the signal-to-noise ratio can be increased, then punctured turbo codes perform very well, tending to have steeper performance curves for a similar signal-to-noise ratio range.

The reduction in performance experienced when removing one third of the codeword prior to transmission is smaller for the SOVA decoder than it is for the Log-MAP decoder, nevertheless, the Log-MAP decoder will reach both voice and data quality bit error rates at a lower signal-to-noise ratio. SOVA will produce the best performance for signal-to-noise ratios below around 1 to 1.5dB where puncturing is used.

With reference to decoder gains, these can be treated in the same way as with un-punctured turbo codes with these frame sizes. The Log-MAP decoder gain increases with signal-to-noise ratio, but the rate at which it increases will be smaller for a code with longer optimal effective free distance than one with a small optimal effective free distance. If voice quality bit error rates are sought, smaller optimised codes will outperform larger optimised codes. This situation is reversed if data quality bit error rates are required.

5.4.3 Results and Analysis of Turbo Codes with 512 Bit Datawords over AWGN

The following results are for longer, 512 bit data frames. These frames are much longer than those previously simulated, however, they may still be referred to as short-frame turbo codes as they are below the guideline of 1000 data bits in length. Having analysed the effects of puncturing in the 100 bit dataword simulations, it was decided that little benefit would come from running simulations both with and without puncturing. Also, as discussed in the introduction, one of the aspects of this

investigation is to evaluate short frame turbo codes in bandwidth constrained conditions for communications that require fast processing and throughput rather than perfect error control. It is likely that in this situation, puncturing would be used.

The same four component codes were simulated, and decoded, using Log-MAP and SOVA decoders for eight decoder iterations except for the 16-state $[23;33]_8$ code. The components of this combination worked very well together and the simulation time necessary to obtain enough bit errors to produce reliable results after more than three iterations was excessive. Therefore, for this combination, results are given for three iterations only. The interleavers were once again chosen using the method described in section 4.4.4. All simulated codes used the puncturing mechanisms described in 4.4.5 and were subjected to AWGN distortion prior to decoding.

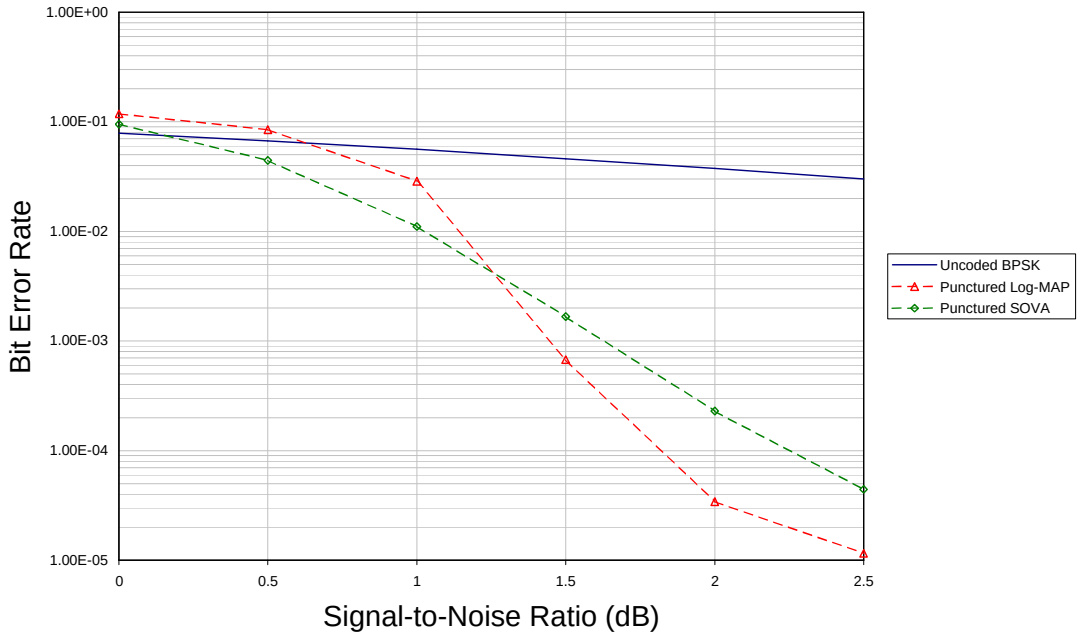


Figure 5.11: SOVA vs. Log-MAP Bit Error Rate for $[7;5]_8$ Component Code over AWGN for 512 Bit Data Frames after 8 Decoder Iterations

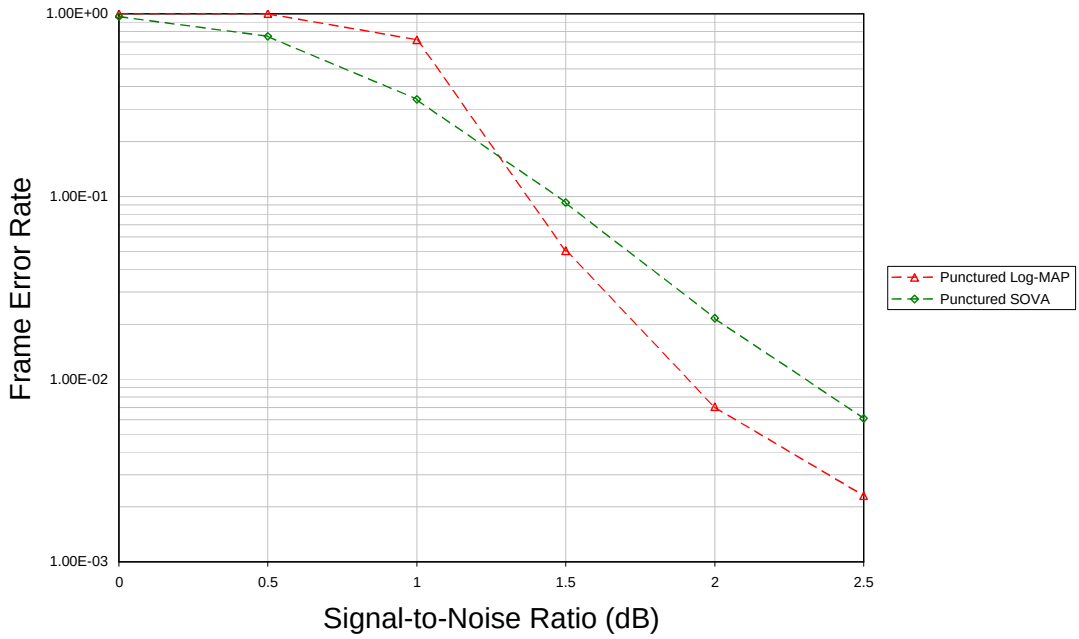


Figure 5.12: SOVA vs. Log-MAP Frame Error Rate for $[7;5]_8$ Component Code over AWGN for 512 Bit Data Frames After 8 Decoder Iterations

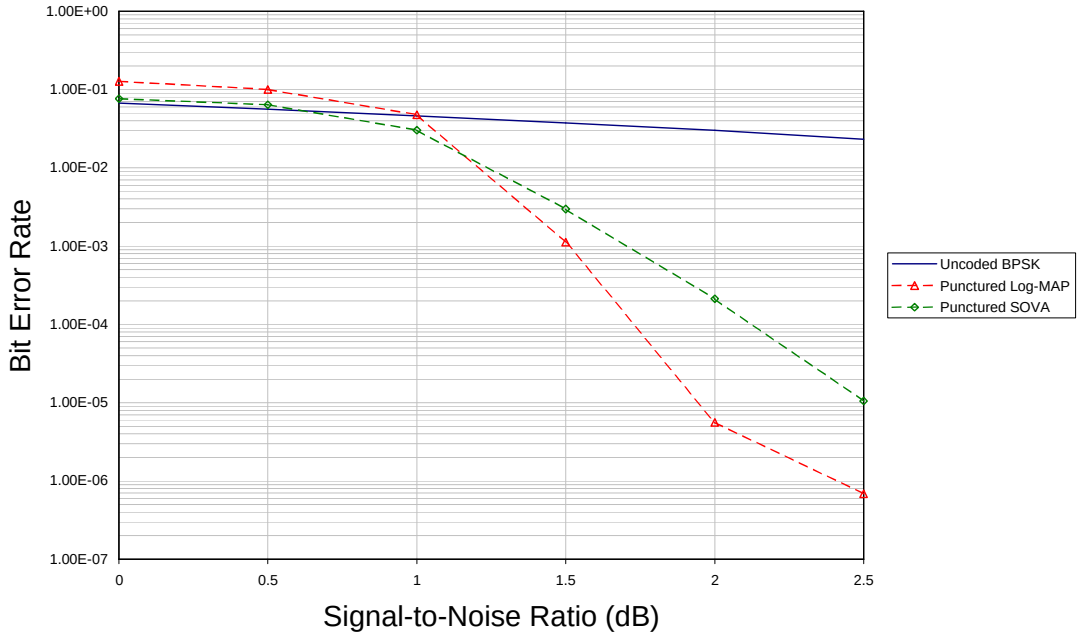


Figure 5.13: SOVA vs. Log-MAP Bit Error Rate for $[15;17]_8$ Component Code over AWGN for 512 Bit Data Frames after 8 Decoder Iterations

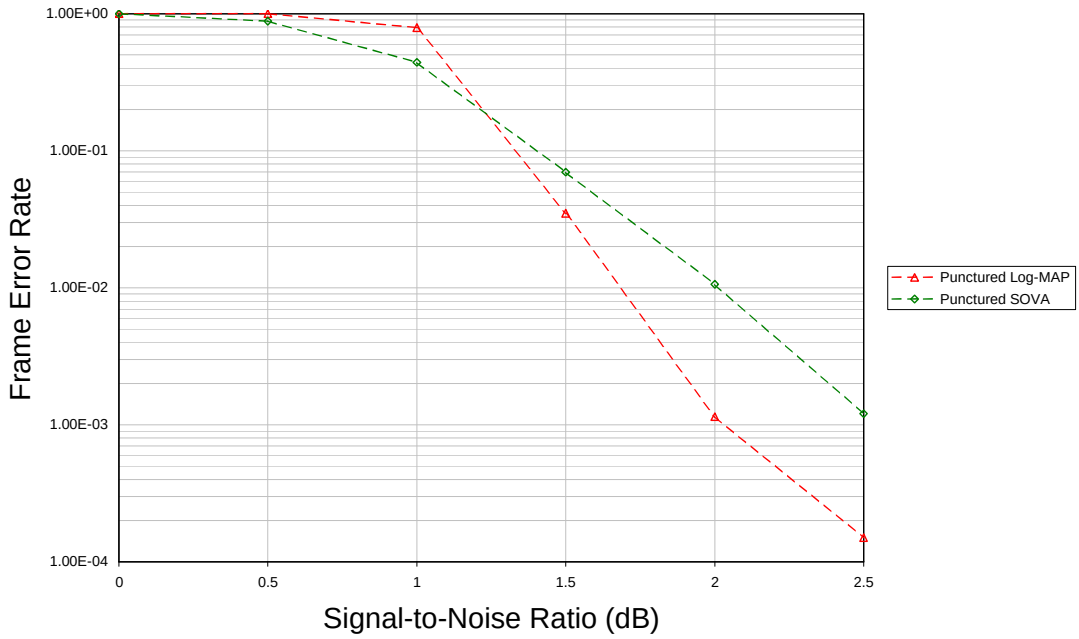


Figure 5.14: SOVA vs. Log-MAP Frame Error Rate for $[15;17]_8$ Component Code over AWGN for 512 Bit Data Frames After 8 Decoder Iterations

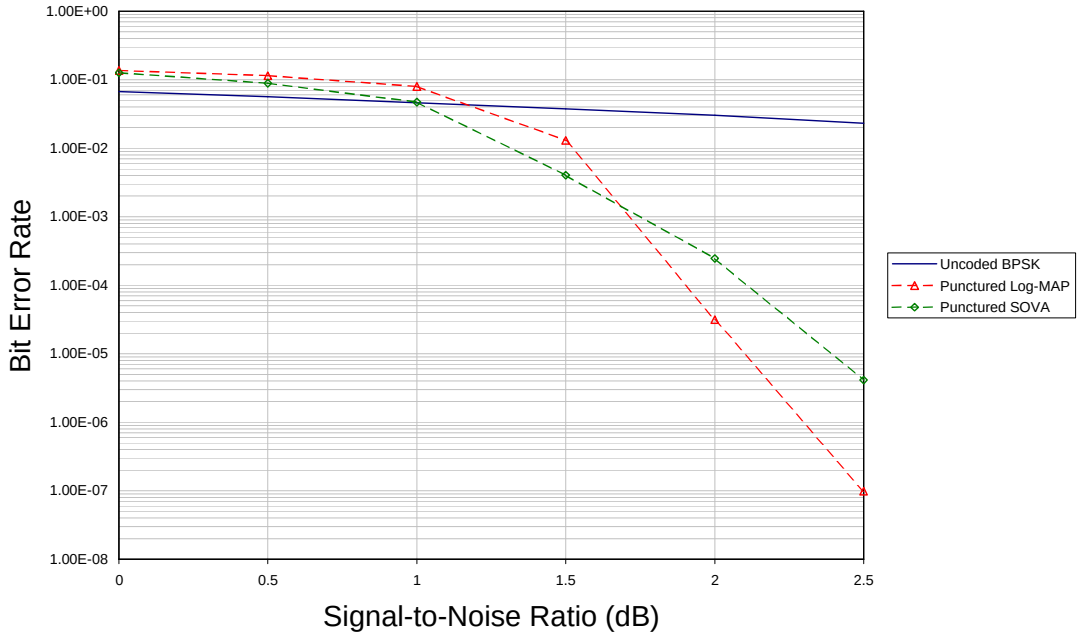


Figure 5.15: SOVA vs. Log-MAP Bit Error Rate for $[23;33]_8$ Component Code over AWGN for 512 Bit Data Frames after 3 Decoder Iterations

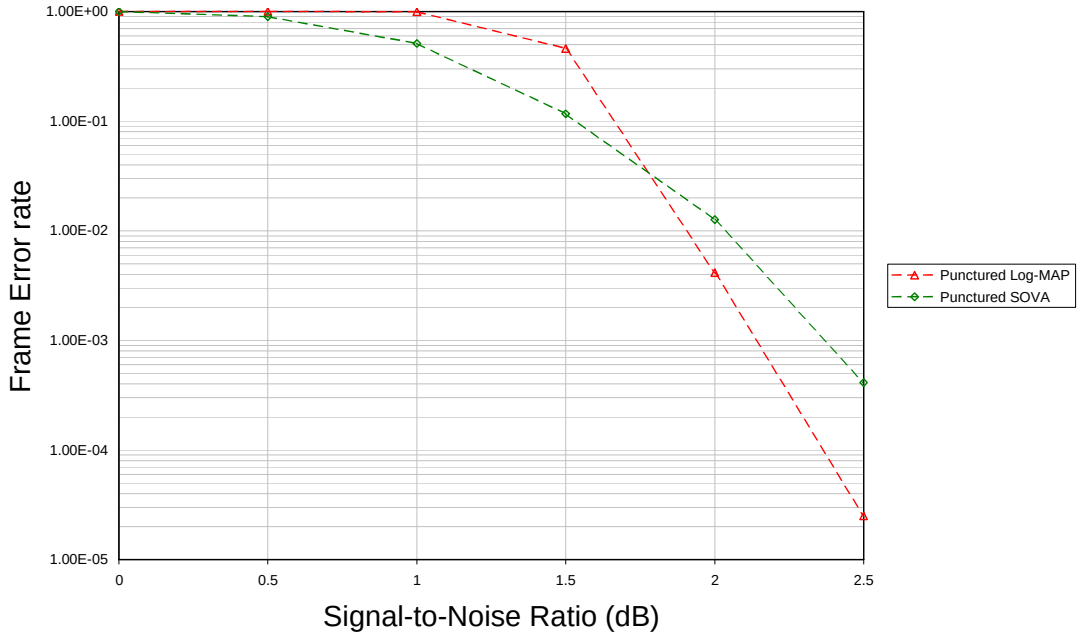


Figure 5.16: SOVA vs. Log-MAP Frame Error Rate for $[23;33]_8$ Component Code over AWGN for 512 Bit Data Frames After 3 Decoder Iterations

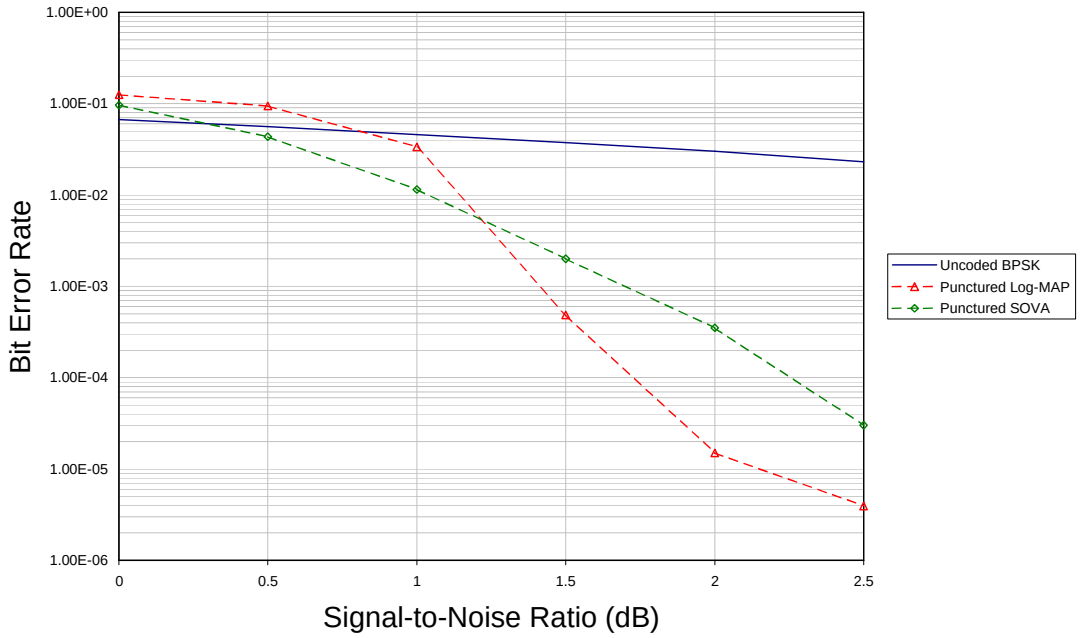


Figure 5.17: SOVA vs. Log-MAP Bit Error Rate for $[63;75]_8$ Component Code over AWGN for 512 Bit Data Frames after 8 Decoder Iterations

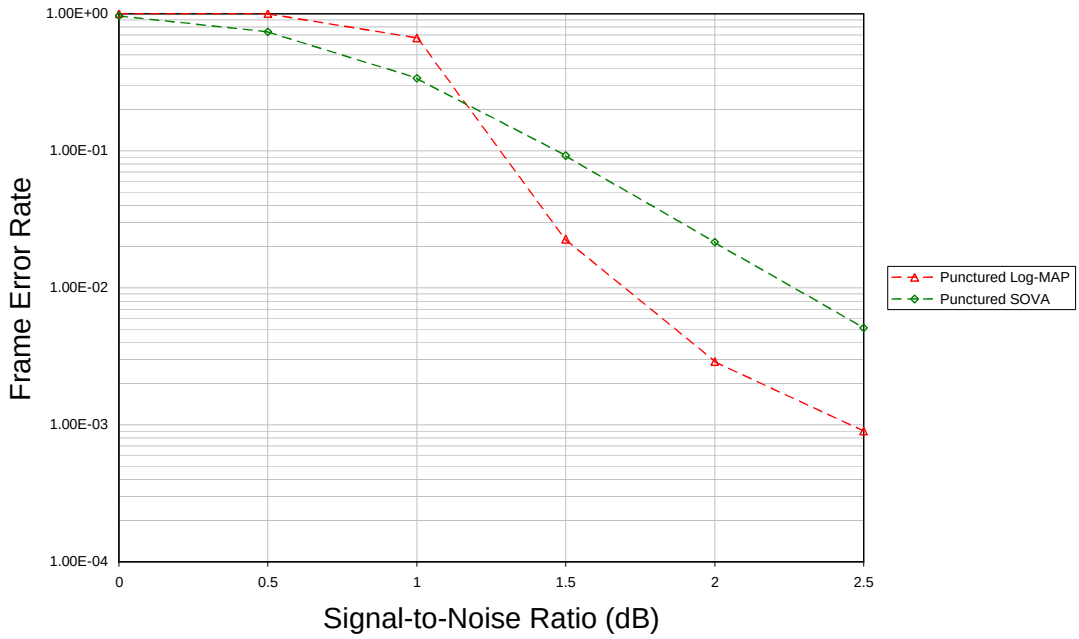


Figure 5.18: SOVA vs. Log-MAP Frame Error Rate for $[63;75]_8$ Component Code over AWGN for 512 Bit Data Frames After 8 Decoder Iterations

Code	Log-MAP 10^{-2}	SOVA 10^{-2}	Log-MAP 10^{-4}	SOVA 10^{-4}
$[7;5]_8$ Punctured	0.308dB	0.542dB	0.87dB	1.09dB
$[15;17]_8$ Punctured	0.287dB	0.45dB	0.894dB	1.334dB
$[23;33]_8$ punctured	0.056dB	0.538dB	0.624dB	1.213dB
$[63;75]_8$ Punctured	0.255dB	0.404dB	0.805dB	0.81dB

Table 5.3: Bit Error Rate SNR Gains for Increase in Data Frame length from 100 bits to 512 bits

Examining the figures and table 5.3, it is immediately obvious that the longer 512 bit dataword simulations outperform those using 100 bit datawords. For both decoders, the gain obtained through increasing dataword length is appreciable. In terms of magnitude, the SOVA results benefit more from the increase in frame length improving results at the voice quality BER by around 0.48dB on average and improving results at 10^{-4} by about 1.113dB on average. This is compared to Log-MAP, which improved by about 0.23dB at the higher BER and around 0.8dB at the lower point of interest.

Code	Log-MAP 10^{-2}	SOVA 10^{-2}	Log-MAP 10^{-4}	SOVA 10^{-4}	Log-MAP BER at 2.5dB	SOVA BER at 2.5dB
$[7;5]_8$ Punctured	1.125dB	1.025dB	1.83dB	2.25dB	$1.16e^{-5}$	$4.43e^{-5}$
$[15;17]_8$ Punctured	1.225dB	1.25dB	1.725dB	2.125dB	$6.88e^{-7}$	$1.05e^{-5}$
$[23;33]_8$ punctured	1.525dB	1.3dB	1.917dB	2.125dB	$9.84e^{-8}$	$4.12e^{-6}$
$[63;75]_8$ Punctured	1.15dB	1.025dB	1.725dB	2.25dB	$3.95e^{-6}$	$3.02e^{-5}$

Table 5.4: Bit Error Rate Observations for 512 Bit Data Frame Simulations

In these simulations, the detrimental effect of choosing a component code without paying attention to whether it is optimal with respect to $d_{free,eff}$ are more obvious. For both decoders, the bit error rate and frame error rate performance curves for the 32-state and the 4-state code are very similar, even though the former code has an effective free distance that is three times that of the latter. The 4-state code has the best possible effective free distance for its constraint length, whereas the 32-state does not.

It is also plain to see that the SOVA decoder outperforms the Log-MAP at low signal-to-noise ratios, reaching a voice quality bit error rate at a lower signal-to-noise ratio than its competitor for all but one code with gains of 0.1dB for the 4-state code, 0.225dB for the 16-state code and 0.125dB for the 32-state code. The 8-state Log-MAP code was the only one to reach a BER of 10^{-2} before the SOVA equivalent with a gain of 0.025dB. At lower bit error rates the decoders revert to the widely accepted situation, with the Log-MAP decoders outclassing their SOVA counterparts with gains of 0.42dB, 0.4dB, 0.208dB and 0.525dB as the constraint lengths increased. The gains made at this point are smaller than those obtained for the 100 bit dataword simulations but do continue to increase with signal-to-noise ratio.

Both the SOVA and Log-MAP algorithms react in the same way when the effective free distance is increased and is optimal for the component code constraint length. Under these circumstances, a larger $d_{free,eff}$ allows the turbo code to reach lower bit error rates. In the simulations, the order of performance for the optimised codes at 2.5dB begins with the $[7;5]_8$, the results of which are improved upon by the $[15;17]_8$ code that is, in turn, improved upon by the $[23;33]_8$ code.

At lower signal to noise ratios however, the order of performance is reversed. In this case, the 4-state code is the most efficient performer, followed by the 8-state code and finally the 16-state code.

It is interesting to note that the $[23;33]_8$ code simulations do not show an error floor, in either the bit error or frame error figures. The only differences between these simulations and those conducted earlier are the dataword length and as a consequence of this, the interleaver construction. The fact that the dataword length increase has not helped with the error floor in the other codes, be they optimised with respect to effective free distance or otherwise, suggests that the interleaver does play an important role in combating this effect. Though it should be noted that fig. 5.1 implies that the error floor effect only begins to become obvious during the third iteration (the number of iterations made with this code)

and it might be supposed that this floor would appear at lower bit error rates. In the 100 bit dataword simulations, the bit error rate at which the error floor came into effect reduced as the constraint length of those codes with optimal $d_{free,eff}$ increased. This appears to be the situation in these simulations when looking at the 4- and 8-state results, in which case the error floor should have appeared at around 3dB for the 16-state code. Although it was not possible to obtain enough errors to continue the simulation of this code, the results that were obtained at this signal-to-noise ratio would, in the worst case, fit an extrapolation that followed the three preceding results along the curve.

With regard to the error floors apparent in the other simulations, it can be see that these effects are more drastic with the longer data frames. This is another problem with the system suggested in [BAR94]. As the size of the interleaver is increased, the strategy becomes increasingly tedious and does not have a definite end (short of testing all possible interleavers). With more recent innovations in interleaver design, a faster solution with improved results is easily attainable [HO98b], [BRE99]. Due to the time required to obtain the results here, it was necessary to conduct multiple simulations concurrently. The error floor effects were therefore not discovered until the results were reviewed.

Table 5.5 shows the signal-to-noise ratios at which the two decoding algorithms reach various frame error rates for each code as well as the performance at 3dB.

Code	Log-MAP 10^{-1}	SOVA 10^{-1}	Log-MAP 6×10^{-3}	SOVA 6×10^{-3}	Log-MAP FER at 3dB	SOVA FER at 3dB
$[7;5]_8$	1.375dB	1.5dB	2.0833dB	2.5dB	2.31×10^{-3}	6.09×10^{-3}
$[15;17]_8$	1.33dB	1.375dB	1.75dB	2.125dB	1.46×10^{-4}	1.22×10^{-3}
$[23;33]_8$	1.667dB	1.525dB	1.95dB	2.11dB	2.5×10^{-5}	4.12×10^{-4}
$[63;75]_8$	1.25dB	1.45dB	1.83dB	2.4375dB	9.04×10^{-4}	5.17×10^{-3}

Table 5.5: Frame Error Rate Observations for 512 Bit Data Frame Simulations

It can be seen from the performance curves above that, as with the bit error rate performance curves, the SOVA decoder outperforms the Log-MAP at low signal-to-noise ratios. For all codes except the 16-state, this effect has disappeared by the time that the frame error rate has reached 10^{-1}. At this point, the Log-MAP codes have begun to outperform the SOVA with gains of 0.125dB for the 4-state code, 0.045dB for the 8-state code and 0.2dB for the 32-state code. The SOVA decoder combined with the 16-state code performs slightly better than the equivalent Log-MAP code with a gain of 0.1417dB at this point of interest.

As mentioned in the bit error rate performance discussion of these codes, the ‘turbo’ effect seems to require increasing signal power to take effect as the constraint length and effective free distances of the optimised codes increase. The frame error performance curves show, in the same way as the bit error curves, that the smaller constraint length codes outperform the higher constraint length codes at lower signal-to-noise ratios, where the codes are optimised. Table 5.5 shows that for the lower frame error rate, the Log-MAP decoding algorithm consistently outperforms its SOVA counterpart

with gains of 0.4167dB, 0.375dB, 0.2675dB and 0.6075dB for the four codes in ascending memory order. It can also be seen that the order of best performance begins with the $[23;33]_8$ code, followed by the $[15;17]_8$ code and the $[7;5]_8$ code. This is compounded by the results at 2.5dB.

Code	Log-MAP 10^{-1}	SOVA 10^{-1}	Log-MAP 6×10^{-3}	SOVA 6×10^{-3}
$[7;5]_8$ Punctured	0.332dB	0.362dB	0.4881dB	0.545dB
$[15;17]_8$ Punctured	0.292dB	0.328dB	0.583dB	0.931dB
$[23;33]_8$ punctured	-0.038dB	0.318dB	0.3dB	0.89dB
$[63;75]_8$ Punctured	0.279dB	0.379dB	0.42dB	0.625dB

Table 5.6: Frame Error Rate SNR Gains for Increase in Data Frame length from 100 bits to 512 bits

As seen in table 5.6, the gains made by increasing the length of the data word by over five times are small. In all but one simulation the increased length of the datawords transmitted improved the performance of the decoders. The only point at which this was not the case was for the 16-state code at a frame error rate of 10^{-1} where the shorter datawords bettered the longer versions by 0.038dB. In general the Log-MAP decoder required around 0.21dB less to achieve a FER of 10^{-1} and 0.44dB less to achieve 6×10^{-3} whereas SOVA required about 0.37dB less for the higher frame error rate and 0.55dB less to retain the lower frame error rate. As with the bit error rate results, the improvement is greater for SOVA, although this is less obvious for the frame error than it is for the bit error.

5.4.4 Conclusions for Turbo Codes with 512 Bit Datawords over AWGN

In conclusion, turbo codes reacted in a similar way to conventional trellis codes in that the increase in the length of the datawords increased the bit error rate performance of the turbo codes. In general, this statement stands for the frame error rate performance results.

The conclusions drawn from the earlier simulations about optimising codes with respect to effective free distance were upheld with these results. In fact, the effects were more obvious. The effective free distance of the 32-state, $[63;75]_8$, code is three times that of the 4-state, $[7;5]_8$, code yet only manages to equal the performance for both decoding algorithms. This highlights the fact that codes should be chosen with optimal effective free distance in mind.

Also more obvious were the error floor effects. The change in dataword length meant that new interleavers were required. As the dataword length increased, so did the number of possible interleavers and the amount of time necessary to test a fair percentage of them to find one that was better than average, using the ‘brute force’ method described in section 4.4.4. Despite this fact and even though a

smaller number of interleavers were tested than for the 100 bit frame simulations, it appears that the interleaver used by the $[23;33]_8$ code was particularly good as no error floor was found.

In comparing the decoders, it was found that the gradient of the performance curves changed in the same way as it did in the earlier simulations, becoming steeper as the effective free distance of the optimised codes increased. It was also found that the SOVA decoder generally reached a voice quality bit error rate at a lower signal-to-noise ratio than Log-MAP. The opposite of this was found for lower bit error rates. The choice of component code was more explicit with these longer codes. Codes with smaller, optimal, effective free distances were found to produce better results at lower signal-to-noise ratios, whereas those with longer optimal effective free distances performed better at high signal-to-noise ratios.

5.4.5 Results and Analysis of Turbo Codes with 100 Bit Datawords over Rayleigh

Following the AWGN simulations, the effect of Rayleigh fading was considered. The two lower constraint length codes were investigated for both decoder strategies, with 100 bit datawords using puncturing to reduce the transmitted code to rate 1/2. Each transmitted bit was subjected to a Rayleigh random variable with variance 0.5 (verified through an iterative process) prior to the addition of white noise.

The effect of insufficient channel fading amplitude estimation was also examined to determine just how important this information is to the decoder and whether one algorithm can cope more comfortably with the lack of information than the other. Perfect fading information was assumed in the simulations that used it (referred to as having Rayleigh reliability in the following results). Where there was no channel state information available to the decoders, AWGN reliability was assumed.

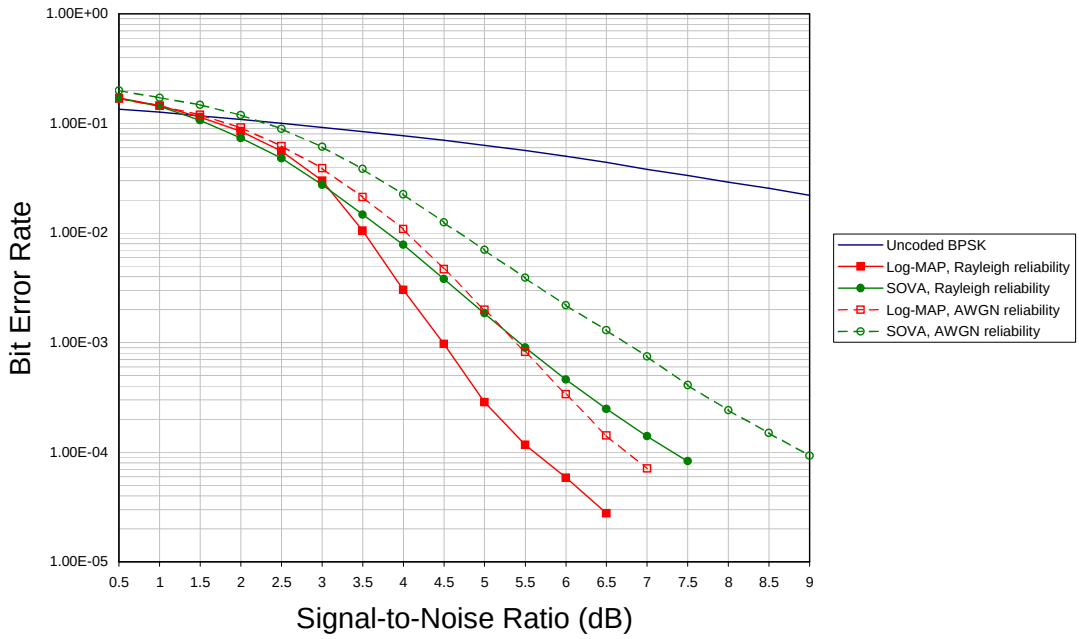


Figure 5.19: SOVA vs Log-MAP Bit Error Rate for $[7;5]_8$ Component Code over Flat Rayleigh Fading Channel with variance 0.5 with 100 bit Data Frames

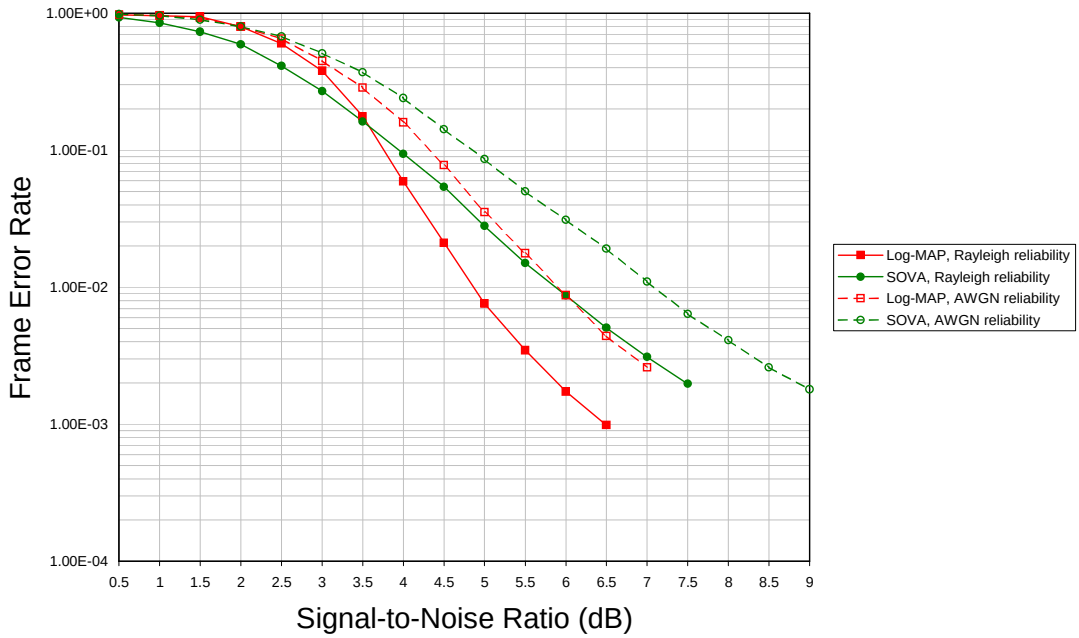


Figure 5.20: SOVA vs Log-MAP Frame Error Rate for $[7;5]_8$ Component Code over Flat Rayleigh Fading Channel with variance 0. 5 with 100 bit Data Frames

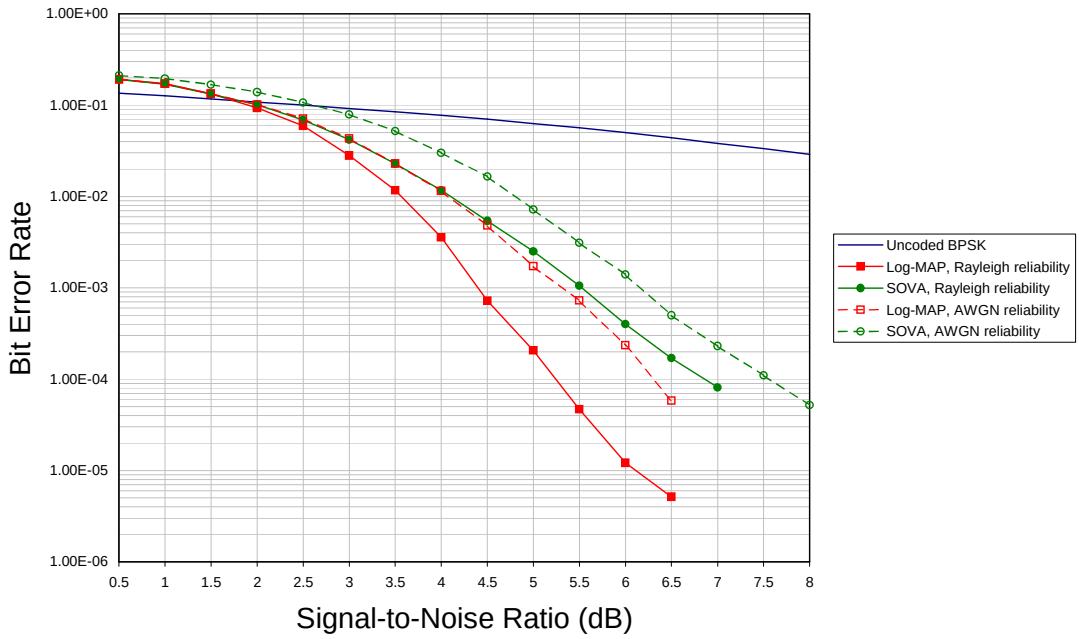


Figure 5.21: SOVA vs Log-MAP Bit Error Rate for $[15;17]_8$ Component Code over Flat Rayleigh Fading Channel with variance 0.5 with 100 bit Data Frames

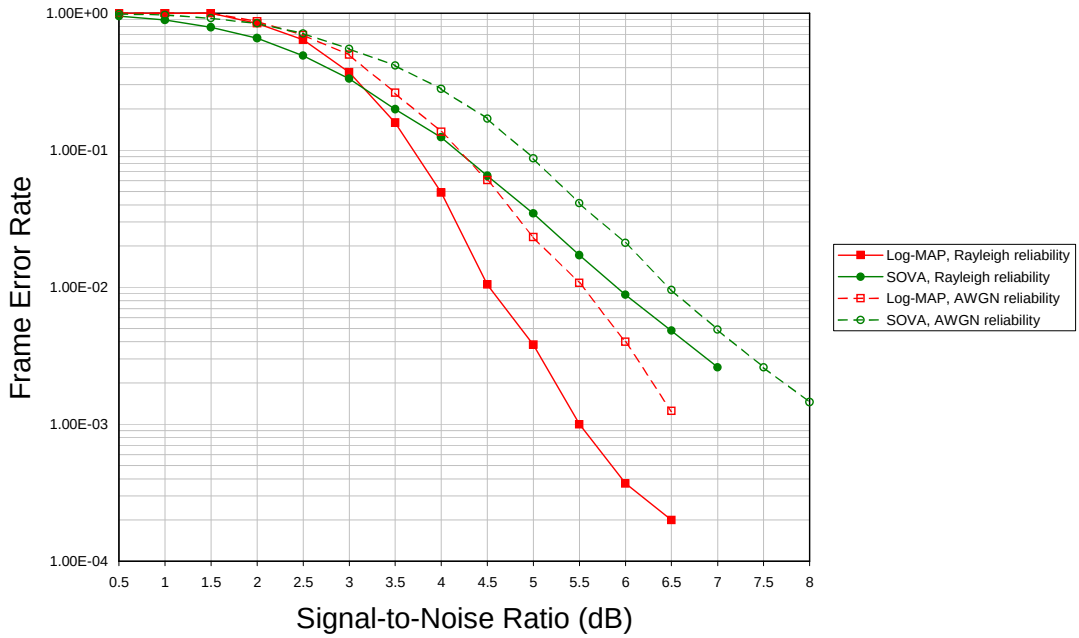


Figure 5.22: SOVA vs Log-MAP Frame Error Rate for $[15;17]_8$ Component Code over Flat Rayleigh Fading Channel with variance 0.5 with 100 bit Data Frames

Table 5.7 shows points of interest for the bit error rate curves of the simulated coding schemes.

	Log-MAP, Rayleigh reliability	SOVA, Rayleigh reliability	Log-MAP, AWGN reliability	SOVA, AWGN reliability
$[7;5]_8$, 10^{-2}	3.5dB	3.75dB	4dB	4.7dB
$[7;5]_8$, 10^{-4}	5.58dB	7.25dB	6.75dB	8.9dB
$[15;17]_8$, 10^{-2}	3.5625dB	4.0625dB	4.0625dB	4.8dB
$[15;17]_8$, 10^{-4}	5.25dB	6.85dB	6.25dB	7.55dB

Table 5.7: Points of Interest in BER performance for Rayleigh Fading Simulations

Looking at the bit error rate results to begin with, there are two immediate observations. First, the Rayleigh multiplicative effect has reduced the performance of the turbo codes with ideal channel amplitude by around 3dB compared with the AWGN results of the same codes, and second, withholding the channel amplitude information completely, reduces the effects of both decoders, although not catastrophically.

Considering the first remark, the shift in performance is to be expected, but it is also evident that the shapes of the performance curves are altered by the multiplicative fading factor.

Although not obvious by inspection, examining the difference between the Rayleigh fading results, where fading information was available to the decoder, and the AWGN results at the two points of interest shows a reduction in the gradient of the bit error rate performance curves. For the $[7;5]_8$ Log-MAP decoder the difference at 10^{-2} is 2.067dB, increasing to 2.88dB at 10^{-4}. The same code, when decoded with SOVA shows a difference of 2.183dB at the higher BER and 3.91dB at the lower. It can be seen that the difference between the two sets of results increases with signal-to-noise ratio and therefore, the gradients of the bit error rate curves of both decoding algorithms are shallower under fading conditions. The same effect is evident in the 8-state code, where the difference between the Log-MAP decoder results for Rayleigh fading with ideal information and the equivalent decoder under AWGN conditions is 2.0505dB at a BER of 10^{-2} and 2.631dB at a BER of 10^{-4}. SOVA reacts in the same way with a difference at 10^{-2} of 2.3625dB and 3.386dB at 10^{-4}. The fact that the difference between the Log-MAP AWGN results and those obtained over Rayleigh fading channels are smaller than the equivalent SOVA results indicates that Log-MAP is less susceptible to this phenomenon than SOVA.

The omission of fading information also affects the performance of these codes. There is a reduction in performance at all levels for both decoders, the magnitude of which increases with signal-to-noise ratio, causing the slopes of the bit error rate performance curves to become shallower still. For the $[7;5]_8$ code in combination with the Log-MAP decoder, the performance at 10^{-2} is degraded by 0.5dB. This increases to 1.17dB at a BER of 10^{-4}. The same code with the SOVA decoder requires a signal-to-noise ratio that is 0.95dB higher with no fading information at 10^{-2}. This grows to a difference of 1.65dB at a BER of 10^{-4}. The situation remains the same with the longer $[15;17]_8$ code. With this code, the Log-MAP simulations with ideal fading information showed a gain over those with no fading information of 0.5dB for a BER of 10^{-2} and 1dB at 10^{-4}. The equivalent SOVA simulations showed

gains of 0.7dB at 10^{-2} and 0.8dB at 10^{-4}. By comparing the results of the Rayleigh simulations with and without fading information at the decoder, it becomes apparent that the performance of the SOVA algorithm is influenced more than the Log-MAP algorithm by the lack of fading information. The gain produced by including fading amplitude at the decoder is generally lower for the Log-MAP decoder than for SOVA. The SOVA decoder does however exhibit a lower gain at the lower bit error rate for the 8-state code.

The results shown in table 5.7 indicate that the two codes perform in similar ways under Rayleigh conditions as they do under AWGN conditions, whether fading information is available or not. In all cases, lower bit error rates were obtained at lower signal-to-noise ratios with the smaller $[7;5]_8$ code and higher bit error rates were obtained at lower bit error rates with the larger $[15;17]_8$ code. For both codes, where fading information was available, SOVA outperformed Log-MAP at the lower signal-to-noise ratios, returning lower bit error rates until approximately 3dB for the $[7;5]_8$ code and around 2dB for the $[15;17]_8$ code. This situation was not repeated in those simulations where fading information was omitted. In these cases, SOVA was consistently outperformed by Log-MAP.

Of course, Log-Map also generally outperformed SOVA in the same way as it did with AWGN. For the $[7;5]_8$ code it produced gains of 0.25dB at 10^{-2} and 1.67dB at 10^{-4} where ideal channel information was available and 0.7dB at 10^{-2} and 2.15dB where it wasn't. These gains follow the AWGN results for the same code, increasing as the signal-to-noise ratio rises, but are significantly larger. The case is the same for the $[15;17]_8$ code, with gains of 0.5dB at 10^{-2} and 1.6dB at 10^{-4} for simulations with ideal information and 0.74dB at 10^{-2} and 1.3dB at 10^{-4} for those without.

The trend is also that the gains made by the log-MAP decoder over its SOVA counterpart are larger when no channel information is available. The exception to this is the lower bit error rate for the longer constraint length code. Here the gain made when ideal fading information was included is 0.3dB larger than that made without, suggesting that the Log-MAP decoding algorithm is more resilient to the effects of channel fading.

	Log-MAP, Rayleigh reliability	SOVA, Rayleigh reliability	Log-MAP, AWGN reliability	SOVA, AWGN reliability
$[7;5]_8$, 10^{-1}	3.75dB	4dB	4.3dB	4.85dB
$[7;5]_8$, 3×10^{-3}	5.55dB	7dB	6.85dB	8.3dB
$[15;17]_8$, 10^{-1}	3.7dB	4.2dB	4.2dB	4.9dB
$[15;17]_8$, 3×10^{-3}	5.6dB	7.37dB	6.62dB	7.87dB

Table 5.8: Points of Interest in FER performance for Rayleigh Fading Simulations

At the higher point of interest, the losses due to the lack of fading information are smaller for Log-MAP than they are for SOVA. The $[7;5]_8$ Log-MAP code, with fading information, shows a

0.55dB gain over the same decoder without fading information at an FER of 10^{-1}, whereas the equivalent SOVA decoder shows a gain of 0.85dB. The $[15;17]_8$ code, with fading information shows a gain at 10^{-1} of 0.5dB for Log-MAP and 0.7dB for SOVA.

At the lower point of interest, the situation is not the same. For the $[7;5]_8$ code, both the SOVA decoder and the Log-MAP decoder show a loss of 1.3dB at 3×10^{-3} and for the $[15;17]_8$ code, the SOVA loss is 0.5dB compared to 1.02dB for log-MAP. These results suggest that, where frame error rate is concerned, the SOVA decoder is affected less by lack of fading information than Log-MAP at lower frame error rates and that at higher frame error rates, Log-MAP is less affected.

Log-MAP consistently outperforms SOVA at the highlighted points of interest as it does in the bit error rate results. However, unlike the bit error results, the frame error rate curves show that SOVA returns better performance at the lower signal-to-noise ratios in simulations where fading information was unavailable and where information was available. For the $[7;5]_8$ code, SOVA was the better performer until around 2dB where fading information was unavailable and 4dB where it was. For the $[15;17]_8$ code, SOVA outperformed Log-MAP until around 2.5dB without fading information and 3.5dB with.

Finally, the order of performance of the two decoders do not match those found with the bit error rate results. At the lower point of interest, 3×10^{-3}, the two decoders show the same performance order as they did when bit error rate was examined, the $[15;17]_8$ code producing better results than the $[7;5]_8$ code at this frame error rate whether fading noise was available or not. The SOVA frame error performance results at the higher point of interest are the same as they were for the bit error results, the smaller code returning a better performance than the longer. Where the orders of performance differ are for the Log-MAP algorithm at the higher point of interest. In the bit error rate results, the order of performance for this decoding algorithm at the higher point of interest was the same as that of SOVA. Where frame error rate is considered, the order of performance is reversed. Here the Log-MAP algorithm produces better performance with the $[15;17]_8$ code than it does with the $[7;5]_8$ version for simulations where fading information was available and without.

5.4.6 Conclusions for Turbo Codes with 100 Bit Datawords over Rayleigh

From the analysis of the Rayleigh simulations, it is plain to see that the effects of fading are two-fold. To begin with, the error rate performance curves are reduced by a substantial amount from those obtained over AWGN. The gradients of the performance curves are also made shallower. The exclusion of fading information affects these decoders as well, further reducing the gradient of the performance curves. The effect is not catastrophic however, and the decoders are still able to counteract the effects of the fading albeit at slightly reduced performance.

The difference between the Log-MAP decoder results for Rayleigh fading where ideal information was supplied and those for AWGN are smaller than the equivalent differences for SOVA.

This suggests that the Log-MAP algorithm is more able to handle Rayleigh effects. Where fading information was not supplied, the bit error rate losses are generally bigger for SOVA, although at the lower bit error rate for the $[15;17]_8$ code, the SOVA loss was less than for Log-MAP.

As before, the design rule, when choosing a code on bit error rate performance depends on the bit error rate required. Smaller codes, with optimal effective free distance, perform better at low signal-to-noise ratios and for higher, voice quality bit error rates, whereas at higher signal-to-noise ratios and for lower bit error rates, the situation is reversed. Where low signal-to-noise ratios are required, SOVA outperforms Log-MAP.

Where frame error rate performance is of more importance, the effects of fading information being available at the decoder are somewhat different. At higher frame error rates, Log-MAP has a smaller loss in performance due to there being no information available at the decoder, whereas, at lower frame error rates, the effect is less detrimental to the SOVA decoder. Whether fading information is available to the decoder or not, the SOVA decoder produces better frame error rates at low signal-to-noise ratios, after which it is consistently outperformed by the Log-MAP algorithm.

For low frame error rates, the choice of code remains as it was for bit error rate performance, the code with the longer $d_{free,eff}$ is better. At high frame error rates, the situation is different. Of the two codes with SOVA decoding, the $[7;5]_8$ code is the better choice, but for Log-MAP, the $[15;17]_8$ code is preferred.

5.5 Turbo Decoder Complexity

When it comes to performance, iteration for iteration, the MAP based decoding algorithms consistently outperform their SOVA counterparts, typically by around 0.5 to 0.7dB, this can be seen above to a lesser extent for shorter frame lengths, but only at lower bit error rates.

Where MAP based algorithms fall down however, is complexity. As stated in the introduction to this thesis, Pietrobon and Barbulescu estimated their modified algorithm to be four times the complexity of the standard Viterbi decoder, while Berrou *et al* calculated SOVA to be around twice the complexity of Viterbi's algorithm.

5.5.1 Complexity Comparisons

Vucetic and Yuan [VUC00] determine the complexity of each decoder according to the following set of rules, calculated as the number of computational operations per time unit for (n,k) component convolutional codes with memory m. It is assumed that Log-MAP is implemented using a look up table for the final `a_post_1` operations as this has been shown to be sufficient, without any loss in performance [ROB95].

Operation	MAP	Log-MAP	Max-Log-MAP	SOVA
Add	$2*2^k*2^m+6$	$6*2^k*2^m+6$	$4*2^k*2^m+8$	$2*2^k*2^m+9$
Multiply	$5*2^k*2^m+8$	2^k*2^m	$2*2^k*2^m$	2^k*2^m
Maximisation		$4*2^m-2$	$4*2^m-2$	$2*2^m-1$
Look up		$4*2^m-2$		
exponential	$2*2^k*2^m$			

Table 5.9: Decoder Complexity Estimates, [VUC00]

Applying this to two rate 1/2 convolutional codes, one with memory m = 2, the other with memory m = 4, the results are:

	k = 2, m = 2			k = 2, m = 4		
Operation	Log-MAP	Max-Log-MAP	SOVA	Log-MAP	Max-Log-MAP	SOVA
Add	102	72	41	390	264	137
Multiply	16	32	16	64	128	64
Max.	14	14	7	62	62	31
Look up	14			62		
Total	**146**	**118**	**64**	**578**	**454**	**232**

Table 5.10: Decoder Complexity Estimate Calculations for Component Codes of Memory m = 2 and m = 4 Following [VUC00]

For these two codes, assuming equal complexity for each operation, it can be seen that Max-Log-MAP is almost twice the complexity of SOVA, while Log-MAP is nearly three times the complexity of SOVA. Although not shown, MAP is very slightly more complex than Log-MAP.

A turbo code system based on MAP family algorithms is therefore at least twice as complex as a turbo code system based on the SOVA decoding algorithm according to [VUC00].

Vucetic and Yuan refer to the [ROB95] in their study of the complexity of the turbo decoding algorithms. This paper does not quite concur with their views on complexity. Table 5.11 below gives estimates as defined by Robertson *et al.*

Operation	Log-MAP	Max-Log-MAP	SOVA
Add	$15*2^m+9$	$10*2^m+11$	$2*2^m+8$
Multiply	8	8	8
Maximisation	$5*2^m-2$	$5*2^m-2$	$3(m+1)+2^m$
Look up	$5*2^m-2$		
Bit compare			$6(m+1)$

Table 5.11: Decoder Complexity Estimates, [ROB95]

Following this set of complexity calculations for the same two codes yields the following:

	K = 2, m = 2			k = 2, m = 4		
Operation	Log-MAP	Max-Log-MAP	SOVA	Log-MAP	Max-Log-MAP	SOVA
Add	69	51	16	489	331	72
Multiply	8	8	8	8	8	8
Max.	18	18	13	158	158	50
Look up	18			158		
Bit Compare			18			36
Total	**113**	**77**	**55**	**413**	**257**	**117**

Table 5.12: Decoder Complexity Estimate Calculations for Component Codes of Memory $m = 2$ and $m = 4$ Following [ROB95]

These estimates show an exponential growth in complexity with memory size of the component code. The Max-Log MAP decoder is only one and a half times the complexity of the SOVA decoder, while the Log-MAP decoder is barely twice as complex for the smaller memory component code, whereas, for the memory $m = 4$ component code, Max-Log-MAP is over twice the complexity of SOVA and the Log-MAP decoder is nearly four times as complex.

Simulations were also undertaken to determine the average time taken for each of the chosen decoding algorithms to process one input frame for a full turbo iteration and varying numbers of states.

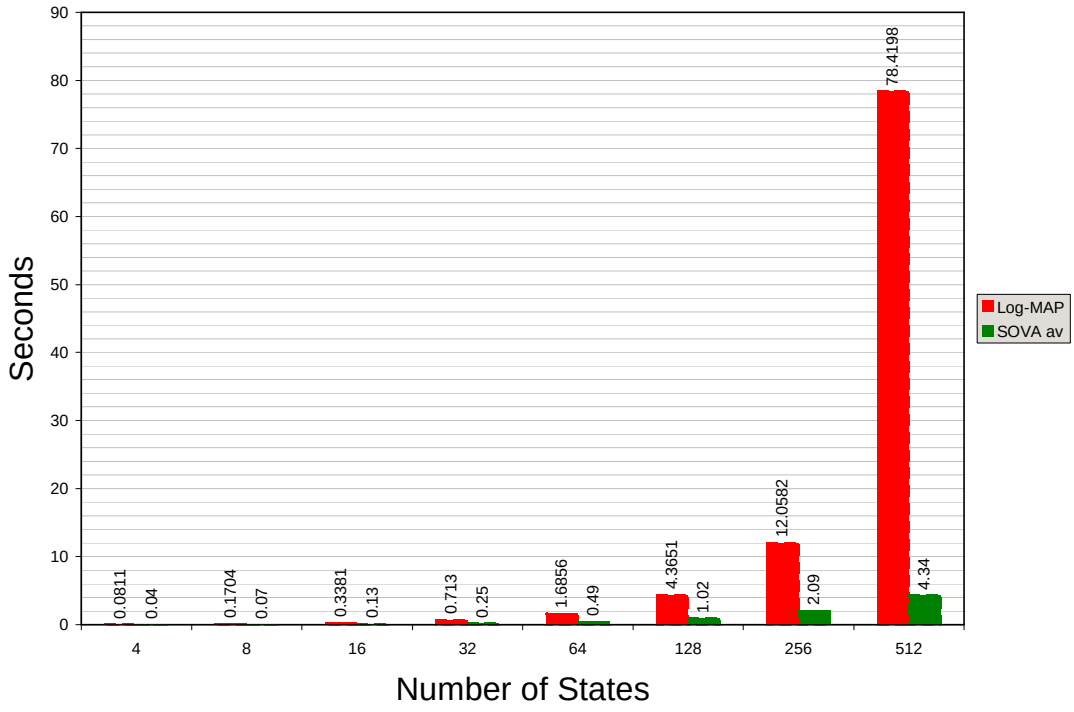


Figure 5.23: Average Time for One Turbo Decode Process Using Log-MAP and SOVA Decoders for Varying Numbers of Trellis States

Figure 5.23 compares the average time taken for one turbo iteration of each of the chosen decoder schemes. The average for each component code was taken over 500 transmitted frames for an increasing number of states. The aim here was not to see how fast the decoder could operate but to review how the increase in complexity of the component codes affected the decoder operations. It can be seen from these results that the time taken for a single Log-MAP decode, for equal frame sizes increases exponentially at a much higher rate, as the number of states in the code trellis increases. Nothing was altered during the simulations, other than the component code constraint length. The results appear to fall somewhere in between the two complexity calculations of [ROB95] and [VUC00], showing an exponential growth unlike [VUC00] but one that is smaller than that forecast by [ROB95].

The balancing of the decoder algorithms, with regard to complexity can therefore be viewed in two ways. The number of iterations permitted for the SOVA decoder can be increased in comparison with the Log-MAP decoder, or the constraint length, and effective free distance, of the SOVA component code can be increased.

Which increase is more beneficial would theoretically depend on the use of the code. Where delay is of little concern and the best possible error performance is the motivation behind using the code, the number of decoder iterations will be high. As seen in figures 5.1 and 5.2, the turbo effect becomes smaller as the number of iterations increase. Therefore, an increase in the number of SOVA iterations may not be beneficial and Log-MAP may still produce better performance. In this case, a SOVA code with higher effective free distance may be more beneficial. It has already been shown that

higher, optimal, effective free distances perform better at low error rates, therefore a SOVA decoder with a much larger optimal $d_{free,eff}$ may produce better results for the same number of iterations, and complexity, as a Log-MAP decoder with smaller optimal component code.

Where the code is to be designed with regard to voice quality bit error rates, a smaller, optimal component code is preferred. In this case, increasing the number of SOVA iterations will not increase decoder delay (another important factor for voice applications), but may well improve performance, especially as SOVA tends to outperform Log-MAP at low signal-to-noise ratios anyway. The increase in constraint length of the component code would be less beneficial here, as the smaller constraint length codes are better performers.

To validate these theories, simulations were undertaken to compare the implemented decoders on a like for like basis with respect to complexity. To investigate the increase in the number of SOVA iterations, a turbo code with a four state $[7;5]_8$ component code was simulated, to be decoded by both SOVA and Log-MAP. The SOVA decoder was run for 6, 8, 12 and 16 iterations, while the Log-MAP decoder was run for 4 and 8 iterations. To accompany this, a turbo code with a 16-state $[23;33]_8$ component code was simulated. The SOVA decoder was run for 6 (less than equivalent according to Vucetic's calculations and below timing results) and 9 (below Robertson's equivalent calculations) iterations, while the Log-MAP decoder was run for 3. Frames were of length 512 data bits and puncturing was used. The results are shown below for AWGN.

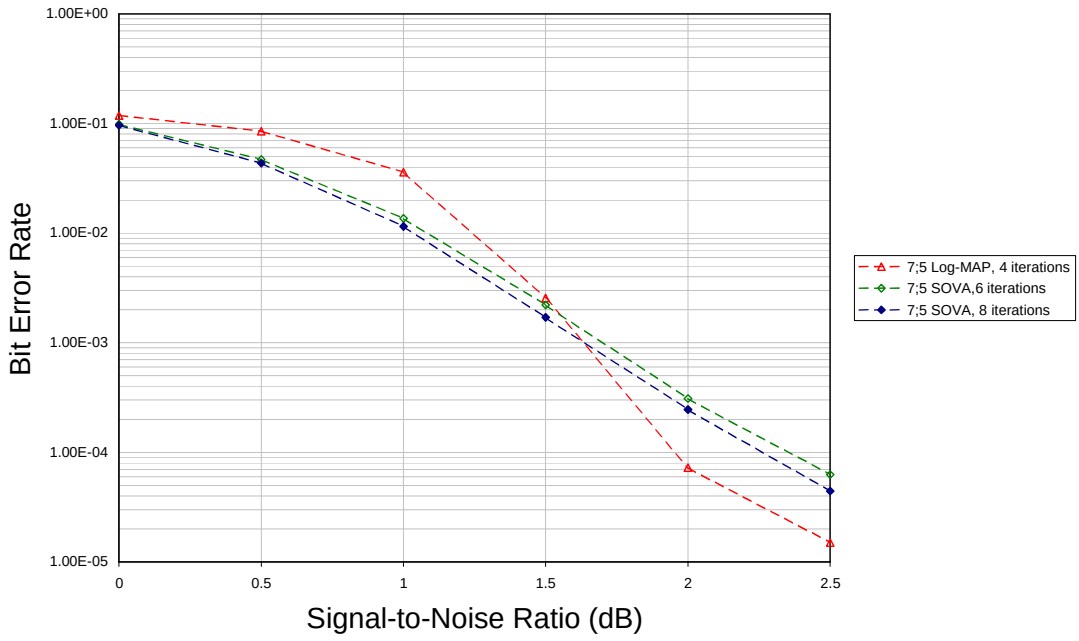


Figure 5.24: SOVA vs. Log-MAP Bit Error Rate over AWGN Channel Based on Complexity Calculations of [VUC00] and [ROB95] for $[7;5]_8$ Component Code

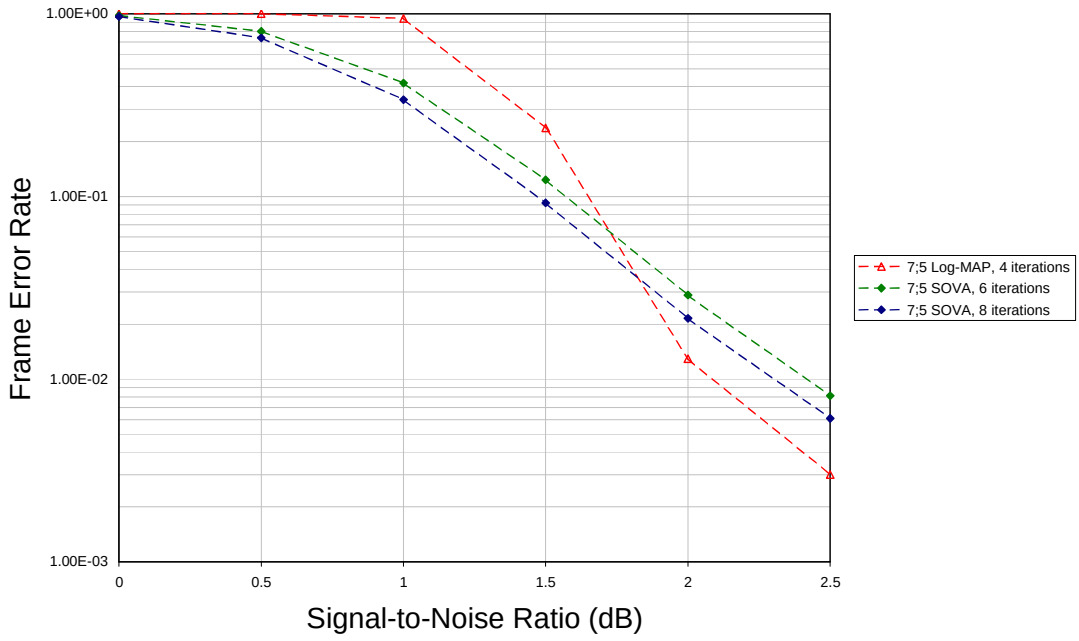


Figure 5.25: SOVA vs. Log-MAP Frame Error Rate over AWGN Channel Based On Complexity Calculations of [VUC00] and [ROB95] for $[7;5]_8$ Component Code

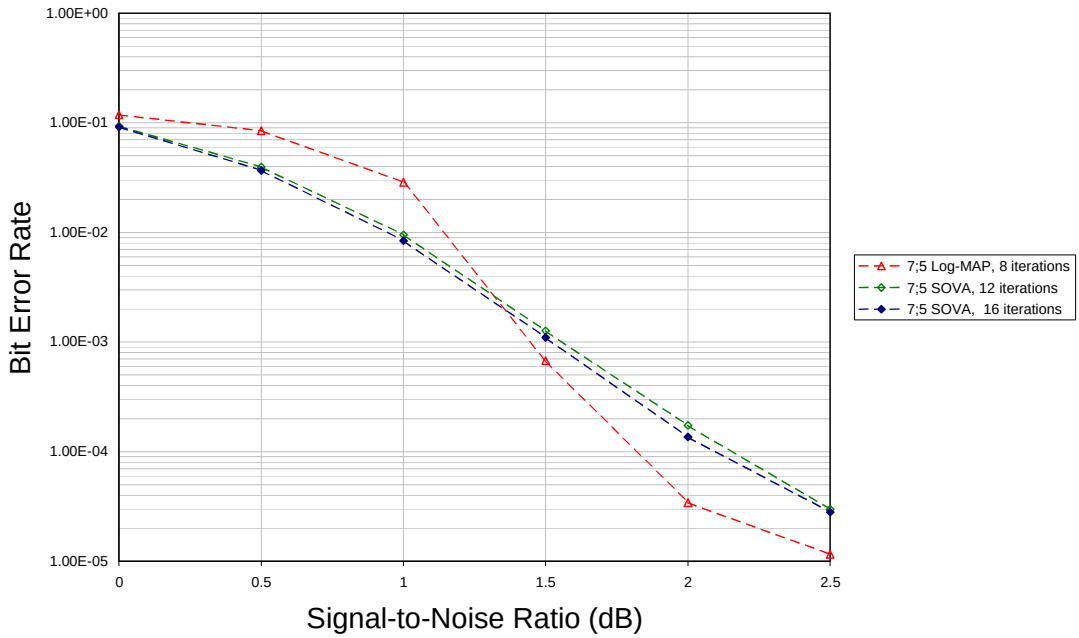


Figure 5.26: SOVA vs. Log-MAP Bit Error Rate over AWGN Channel Based on Complexity Calculations of [VUC00] and [ROB95] for $[7;5]_8$ Component Code

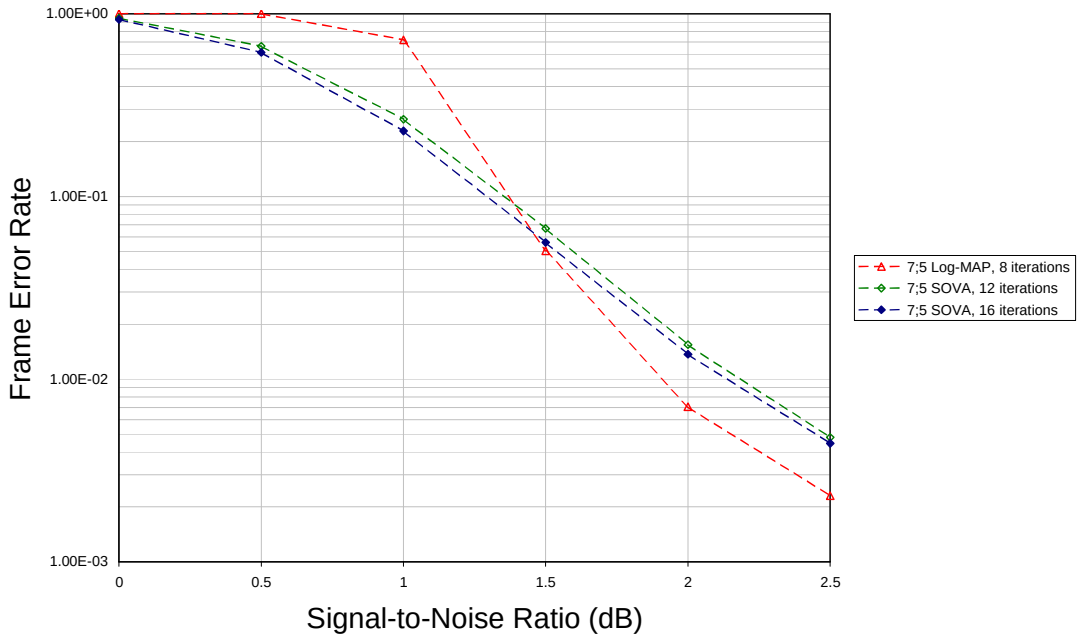


Figure 5.27: SOVA vs. Log-MAP Bit Error Rate over AWGN Channel Based on Complexity Calculations of [VUC00] and [ROB95] for $[7;5]_8$ Component Code

With these results it is possible to compare the decoding algorithms on a complexity basis, and therefore also on a decoding delay basis. The complexity calculations of both [ROB95] and [VUC00] as well as the results of simulation timing averages concur for the $[7;5]_8$ code that the SOVA decoder has a little less than half the complexity of the Log-MAP decoder.

Table 5.13 below shows some points of observation obtained from these simulations

Code, iterations	BER = 10^{-2}	BER = 10^{-4}	FER = 10^{-1}	FER = $8x10^{-3}$
Log-MAP, 4	1.25dB	1.95dB	1.65dB	2.16dB
SOVA, 6	1.07dB	2.33dB	1.56dB	2.5dB
SOVA, 8	1dB	2.25dB	1.46dB	2.4dB
Log-MAP, 8	1.125dB	1.81dB	1.375dB	1.95dB
SOVA, 12	0.99dB	2.125dB	1.37dB	2.275dB
SOVA, 16	0.92dB	2.08dB	1.3dB	2.21dB

Table 5.13: Points of Observation for $[7;5]_8$ Complexity Simulations

The comparison between 4 Log-MAP iterations and 6 or 8 SOVA iterations for a BER of 10^{-2} yields performance gains of 0.18dB and 0.25dB respectively and the comparison between 8 Log-MAP iterations and 12 or 16 SOVA iterations yields gains of 0.135dB and 0.205dB respectively. This shows that, depending on which complexity calculations are correct, a SOVA turbo decoder with the same

component code and equivalent, or less, decoder complexity will perform better at voice quality bit error rates than a Log-MAP decoder will.

However, lower BERs remain more easily obtainable with the Log-MAP algorithm. The Log-MAP decoder after 4 iterations performed better than the SOVA at 6 or 8 iterations, with performance gains of 0.38dB and 0.3dB respectively. The same can be said for the 8-iteration Log-MAP decoder, producing gains of 0.315dB and 0.27dB over the SOVA 12 and 16 iteration simulations respectively.

At the two frame error comparison points, the same applies. SOVA outperformed Log-MAP in the both sets of simulations, with gains of 0.09dB and 0.19dB for the 6 and 8 iteration SOVA decoders and gains of 0.005dB for the 12 iteration SOVA decoder and 0.075dB for the 16 iteration decoder.

At the lower FER, Log-MAP produced gains for both sets of simulations, beating SOVA by 0.34dB for 6 SOVA iterations and 0.24dB for 8 SOVA iterations, compared with 4 Log-MAP iterations and 0.325dB and 0.26dB for the 8-iteration complexity comparison.

Similar simulations were then carried out for the 16-state $[23;33]_8$ code, although only comparing 3 decoder iterations for the Log-MAP decoder as attaining a satisfactory bit error rate for higher iterations proved excessively time consuming. The results are shown below.

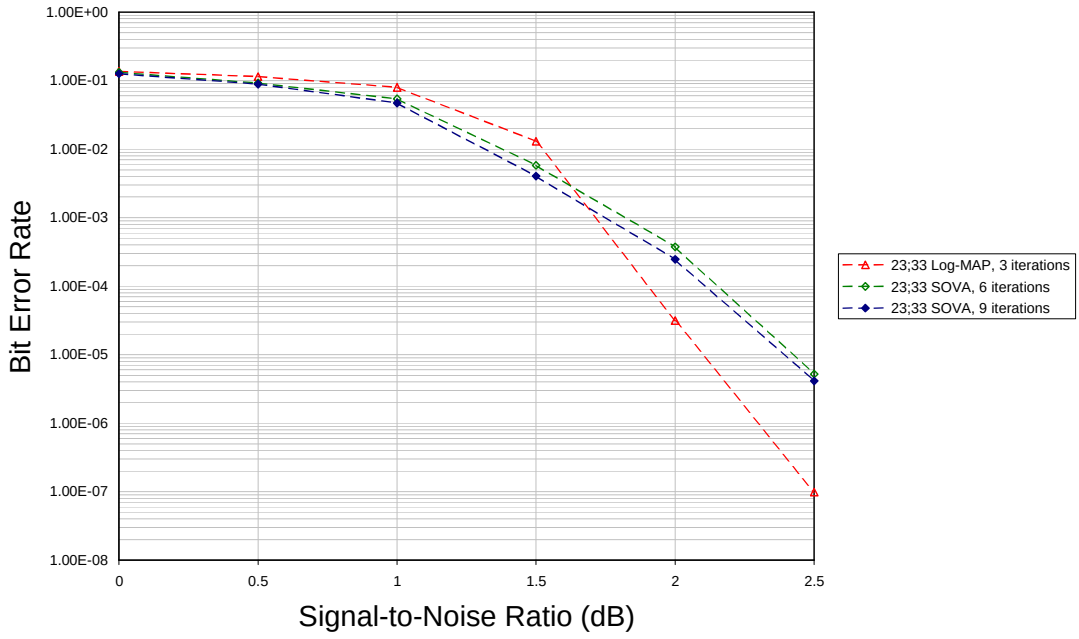


Figure 5.28: SOVA vs. Log-MAP Bit Error Rate over AWGN Channel Based on Complexity Calculations of [VUC00] and [ROB95] for $[23;33]_8$ Component Code

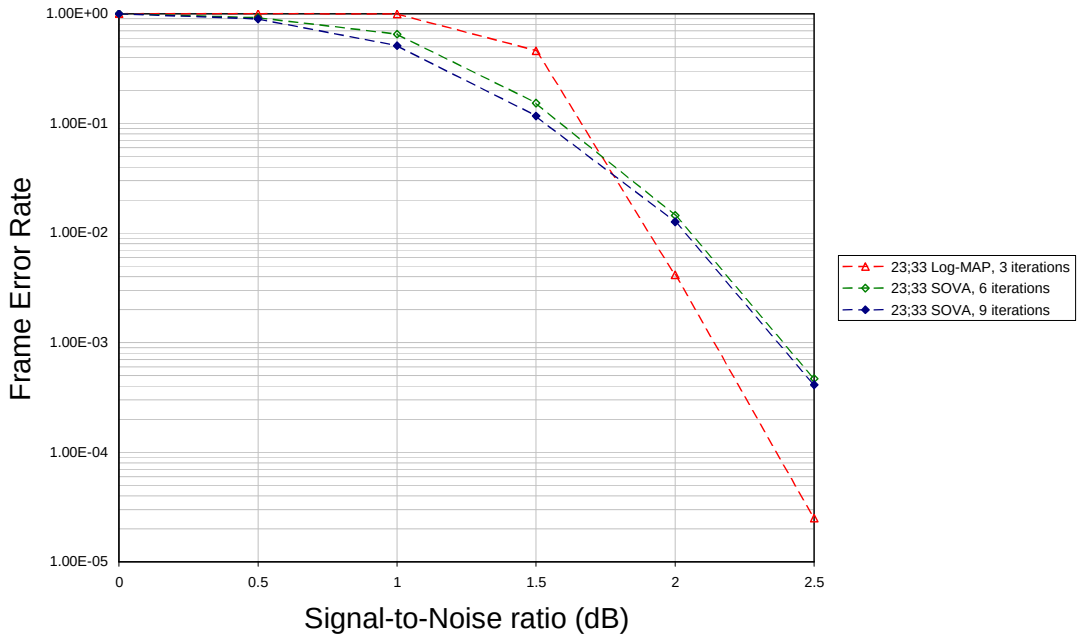


Figure 5.29: SOVA vs. Log-MAP Frame Error Rate over AWGN Channel Based On Complexity Calculations of [VUC00] and [ROB95] for $[23;33]_8$ Component Code

As can be seen in figures 5.28 and 5.29, the results are very similar to those discussed earlier. Table 5.14 below highlights the points of interest along the two graphs.

Code, iterations	BER = 10^{-2}	BER = 10^{-4}	FER = 10^{-1}	FER = 10^{-3}
Log-MAP, 3	1.5dB	1.9dB	1.64dB	2.14dB
SOVA, 6	1.36dB	2.14dB	1.57dB	2.4dB
SOVA, 9	1.3dB	2.08dB	1.52dB	2.375dB

Table 5.14: Points of Interest in Decoder Complexity Comparisons for $[23;33]_8$ Component Code

As before, it can be seen that the SOVA decoder reaches a voice quality bit error rate at a lower signal-to-noise ratio than the Log-MAP decoder does, whether the decoder complexity is equal or less. The SOVA decoders create a gain of 0.14dB for 6 decoder iterations, rising to 0.2dB for 9 decoder iterations when the desired BER is 10^{-2}. Also as before however, the situation is reversed for lower bit error rate targets and Log-MAP improves upon the results of SOVA at 10^{-4} with gains of 0.24dB and 0.18dB over the two SOVA schemes. This all remains unchanged when considering the frame error rates with SOVA reaching an FER of 10^{-1} before Log-MAP with gains of 0.07dB and 0.12dB and Log-MAP beating the two SOVA schemes to an FER of 10^{-3} with gains of 0.26dB over the 6 iteration scheme and 0.235dB over the 9 iteration scheme.

The complexity of the SOVA decoder can also be made equal to the Log-MAP decoder through increasing the effective free distance of the component code. This may be beneficial where the

error rate performance is the overriding factor and decoder delay is less important. According to Vucetic, the $[23;33]_8$ SOVA decoder is 1.5 times the complexity of the $[7;5]_8$ Log-MAP decoder, the timing information concurs with this. Robertson estimates the two codes to be equal in complexity. Figures 5.38 and 5.39 compare these two decoder-code set-ups for bit error and frame error performance in AWGN conditions. Datawords were set to 512 bits, puncturing was used and eight decoder iterations were performed for each decoder.

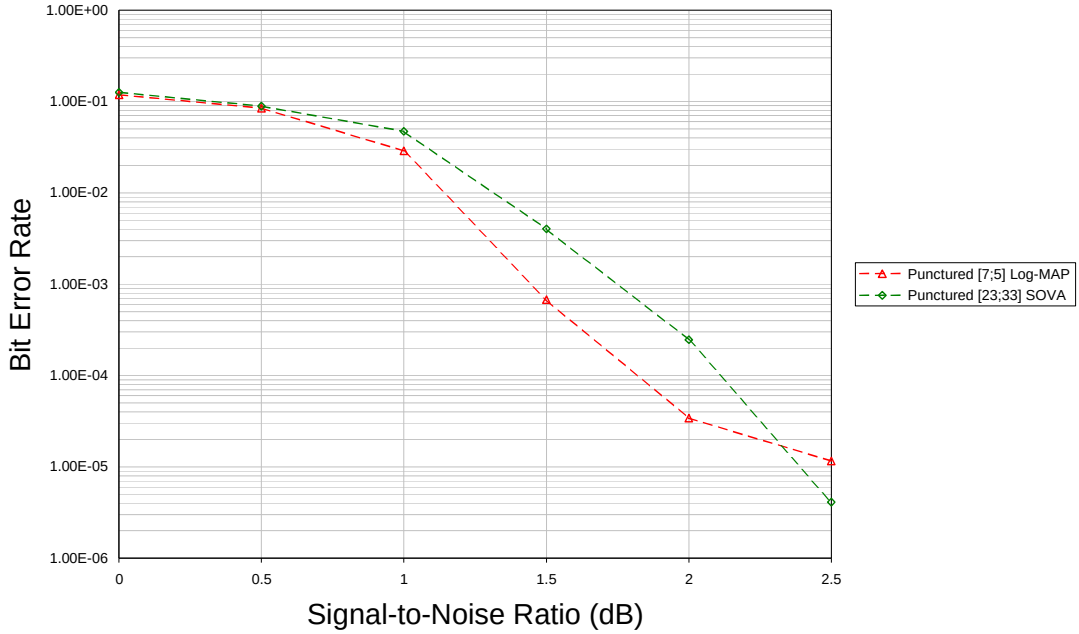


Figure 5.30: $[7;5]_8$ Log-MAP vs. $[23;33]_8$ SOVA Bit Error Rate over AWGN Channel for 8 iterations

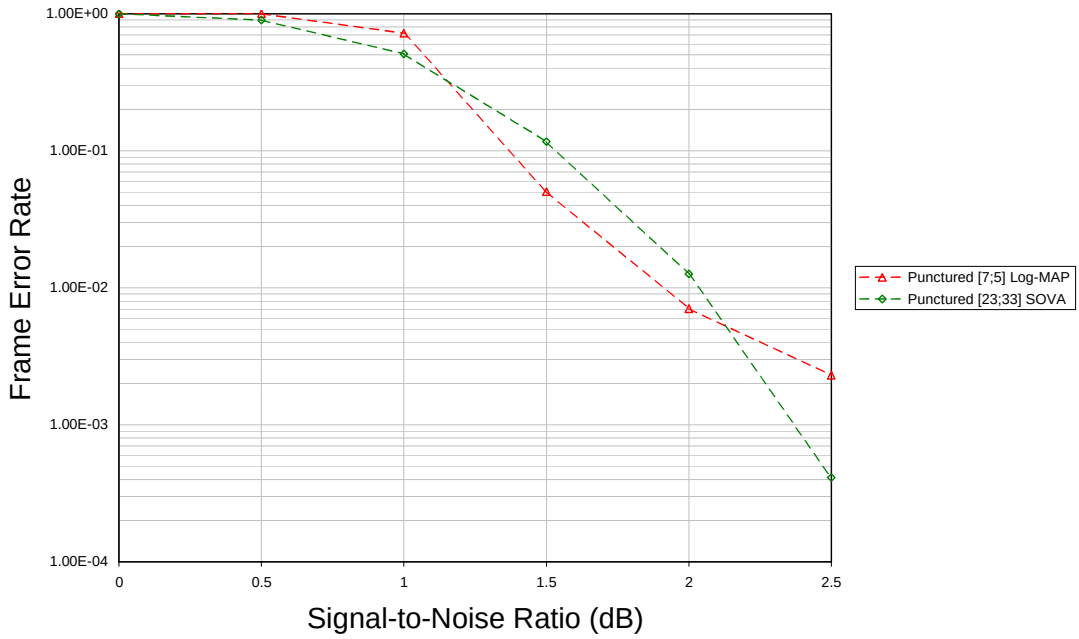


Figure 5.31: $[7;5]_8$ Log-MAP vs. $[23;33]_8$ SOVA Frame Error Rate over AWGN Channel for 8 iterations

These two figures show what was already determined earlier, the code with the lower optimal effective free distance outperforms that with the higher optimal effective free distance at lower signal-to-noise ratios, but the situation is reversed as the signal-to-noise ratio increases. However, now these codes can be referred to as equivalent (or nearly equivalent) in terms of complexity, which would make SOVA the better performer. It should be noted that for the original 100 bit dataword AWGN simulations for the same codes, the SOVA decoder did not outperform the Log-MAP at higher signal to noise ratios.

5.5.2 Complexity Conclusions

This section investigated the complexity of the two competing decoder algorithms. It was found that the SOVA algorithm was much less complex than the Log-MAP algorithm. Results were therefore obtained for the two decoding algorithms on an equal complexity basis.

The two decoders were compared for the same component codes, with different numbers of decoder iterations and it was found that in these cases SOVA required less signal-to-noise ratio to attain voice quality performance with equal or less complexity than Log-MAP. Log-MAP still outperformed SOVA at lower performance rates.

According to the timing data, four iterations of the Log-MAP turbo decoder with the 4-state code are equivalent to 8 SOVA decoder iterations. The SOVA decoder outperformed Log-MAP after 6 decoder operations, reducing the decoder delay substantially. The timing data for the 16 state code

shows that 3 Log-MAP decoder iterations were roughly equivalent to 9 SOVA decoder iterations. SOVA outperformed Log-MAP after 6 iterations.

The two decoding algorithms could also be compared with equal numbers of iterations if the component codes were different. It was shown earlier that higher optimal effective free distances produced better results at higher signal-to-noise ratios and it was therefore unsurprising that the SOVA decoder with the 16-state code outperformed the Log-MAP decoder using the 4-state code at higher signal-to-noise ratios. This did only occur under specific circumstances, where the simulations used 512 bit datawords. For smaller datawords, the Log-MAP code with smaller effective free distance still outperformed the SOVA decoder with larger effective free distance.

5.6 SOVA Susceptibility to SNR Estimation Errors

As mentioned in section 2.5, research has shown that perfect SNR estimation is not critical for the Log-MAP and Max-Log-MAP decoding algorithms when used in turbo codes. [WOR00] showed that Max-Log-MAP was theoretically independent of the SNR and that errors in the estimated SNR produced a negligible effect in Log-MAP.

The authors showed that a single fixed SNR estimate that was within 1.5dB of the actual SNR had very little effect on Log-MAP simulations over a range of actual signal-to-noise ratios and also investigated the effects of offset estimates at a particular signal-to-noise ratio, comparing estimates that were offset from the correct SNR by $\pm$4dB. It was noted by the authors that the correct SNR estimate did not produce the best performance and that, in fact, the best results were obtained when the estimate was less than the correct value. This peculiarity was only found in short frame simulations (the authors used 600 data bits) and the authors stated that results for codes using longer frames did not exhibit this. According to the results of [WOR00], bit error performance was only severely reduced when the signal-to-noise ratio estimate was 4dB below the actual value. At this point, the discrepancies in results from those obtained with correct SNR ranged from over 2 orders of magnitude at 1dB to 1 order of magnitude at 2dB. Estimates offset by less than 4dB were all within half an order of magnitude of the results obtained with the correct value. The authors also concluded from the results that it was preferable to overestimate the signal-to-noise ratio than underestimate, as the results with overestimation did not drastically change.

If the case is to be made that SOVA may be more suited to small data frame transmission systems than Log-MAP, it is necessary to investigate the effects of errors in SNR estimation. This has never been published for the SOVA decoding algorithm. Using similar code specifications and methods to those of [WOR00], this section shows the effects of a constant error in SNR estimation during a SOVA transmission and a variety of errors in estimation for particular transmitted signal-to-noise ratios.

The following two figures illustrate the effect of overestimating and underestimating the SNR at the decoder. The simulated turbo code used the [7;5] RSC component codes and the interleaver

determined earlier in the chapter for 100 bit data frames. The output codewords were left un-punctured and subjected to AWGN before undergoing 8 decoder iterations. The figures below show the effect of underestimating the SNR by 3dB and overestimating the SNR by 3dB and show this in comparison with a third curve showing results obtained with the correct SNR value.

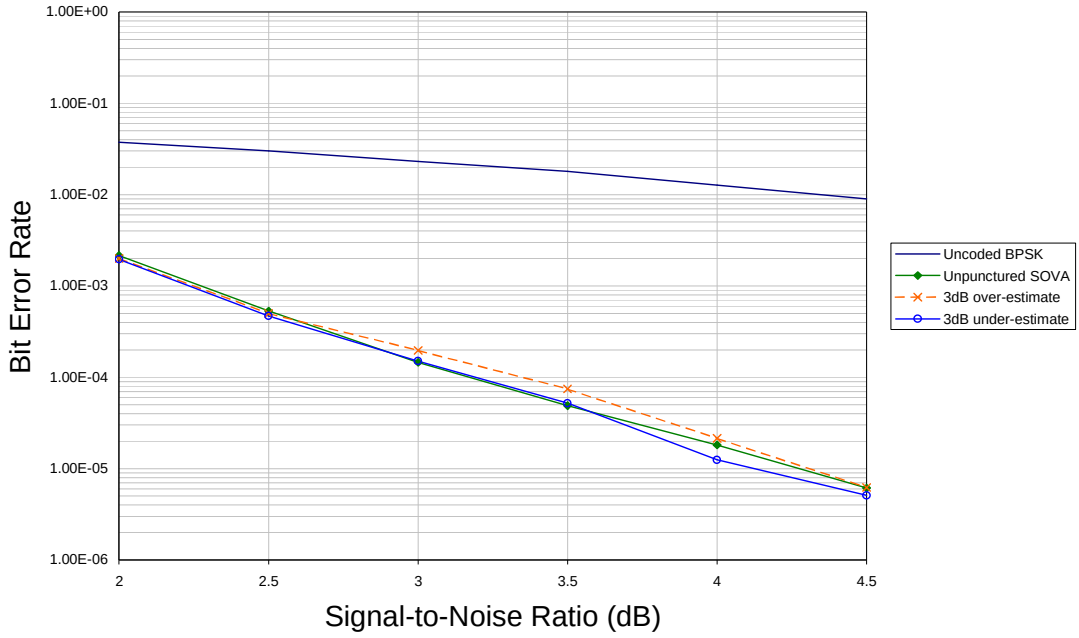


Figure 5.32: The Effect on BER of Inaccurate SNR Estimation on SOVA Turbo Decoding over AWGN Channel

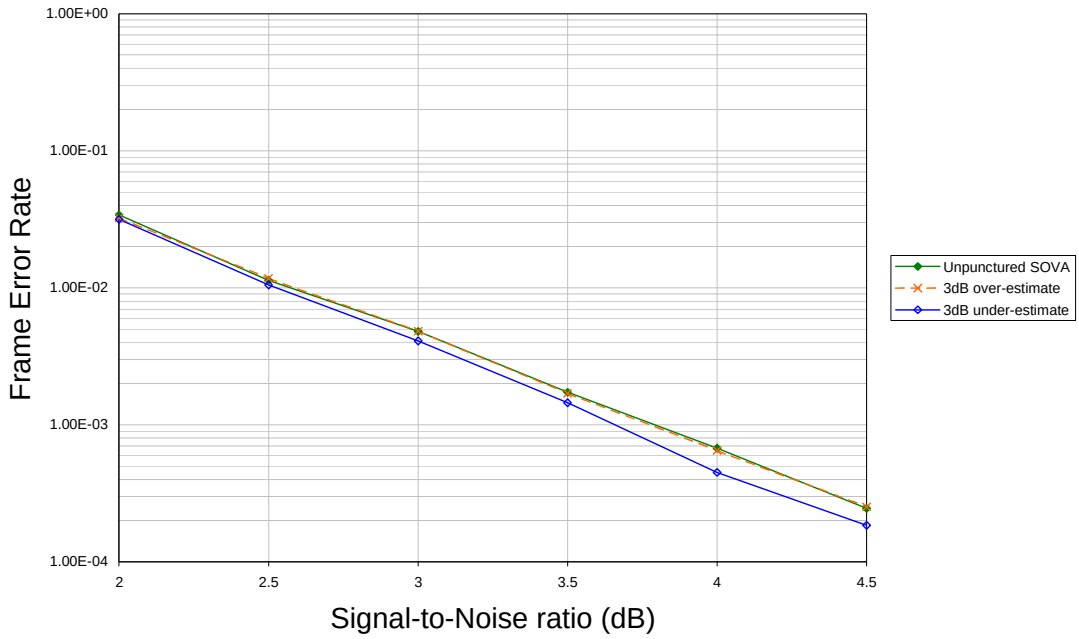


Figure 5.33: The Effect on FER of Inaccurate SNR Estimation on SOVA Turbo Decoding over AWGN Channel

The results above are similar to those of [WOR00] in that a pessimistic estimate of the transmitted SNR produces improved results upon those obtained with the correct SNR value. It is also apparent that an optimistic estimate of this size is not detrimental to the performance of the code. This shows that the SOVA decoder, like its more complex counterpart, is quite resilient to errors in SNR estimation.

The following two figures illustrate the effect of various SNR estimate errors on the same turbo code at 0dB, 2dB and 3dB actual SNR.

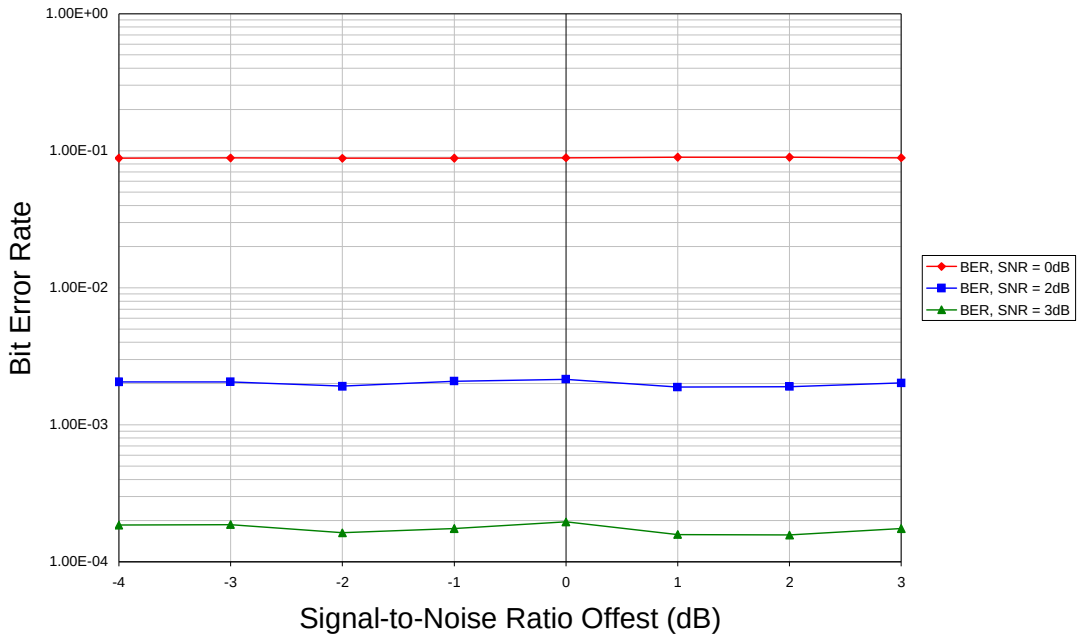


Figure 5.34: Bit Error Rate Performances for Offset Signal-to-Noise Ratio Estimates over AWGN Channel

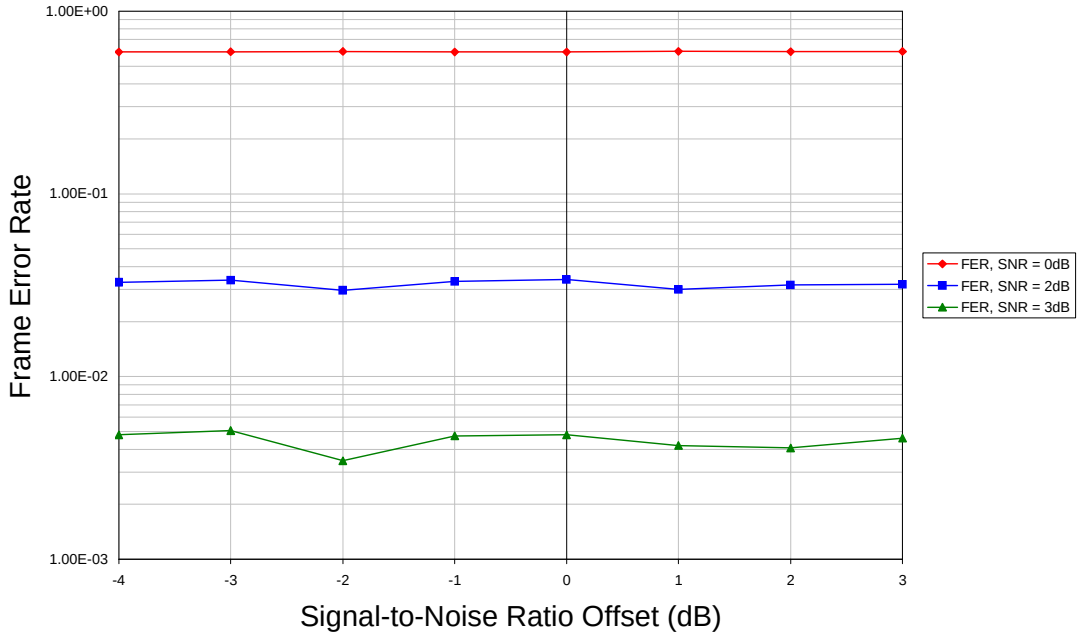


Figure 5.35: Frame Error Rate Performances for Offset Signal-to-Noise Ratio Estimates over AWGN Channel

Figures 5.34 and 5.35 show that the SOVA decoding algorithm is affected only slightly by errors in signal-to-noise ratio estimates. The decoder at a transmitted signal-to-noise ratio of 0dB actually produced the best performance with a pessimistic estimate of –2dB. The same decoder

produced the best performance for a transmitted signal-to-noise ratio of 2dB with an optimistic estimate of 3dB and for a transmitted SNR of 3dB, produced the best results with an estimate of 5dB.

As with the bit error results, the frame error results imply that the SOVA decoder is resistant to errors in SNR estimation. Even with an estimate that is 4dB below actual, the decoder is not significantly affected.

While the mathematical derivation of this algorithm in chapter 4 shows that an estimate of the SNR is necessary for decoding, it does not show how much effect this variable has on the result. As stated earlier, the research of [WOR00] showed that the Log-MAP decoder could comfortably function with as much as $\pm$3dB error in the SNR estimate, but that larger errors could cause catastrophic effects. The results produced here for the SOVA decoder in figures 5.34 and 5.35 show that this algorithm is more resilient to these errors. Where the Log-MAP algorithm would fail, with errors in SNR estimation of –4dB, SOVA is comfortably producing results very similar to those produced with perfect knowledge of transmitted SNR. What does become apparent from 5.34 and 5.35, is that the sensitivity of the SOVA decoder to these estimation errors increases with transmission SNR.

5.7 Conclusion

This chapter outlines the design and modular software implementation of a generic turbo decoder system for SOVA and Log-MAP decoding. A fully worked example of both the SOVA decoding algorithm and its Log-MAP counterpart using real received values has also been included. The worked example has been included as an aid to further understanding and visualisation of the turbo decoding concept as literature has been found to be lacking such a full description. The worked example process helps to highlight the similarities between the two decoding algorithms as well as their differences, it also brings to light the variations in complexity.

Investigations were then made into the performance of short frame turbo codes over AWGN channels. Four component codes with increasing constraint lengths were considered, two commonly used in the literature ($[7;5]_8$ and $[15;17]_8$) and having maximum effective free distance for their constraint lengths and rates, one previously unpublished but also displaying optimal effective free distance for its parameters ($[23;33]_8$) and a fourth that did not have an optimal effective free distance ($[63;75]_8$).

Results in the form of bit error rate performance and frame error rate performance for all four codes were obtained over AWGN channels for data frames of 100 with puncturing and without.

As expected, the codes that included complete sets of both parity streams outperformed those that were punctured. It was found that error floors were less apparent for these un-punctured codes and that this was probably connected to the interleaver design criteria used in this project.

The gains made with the Log-MAP decoder over its SOVA counterpart increased with signal-to-noise ratio and optimal effective free distance and were larger for un-punctured codes than punctured, which suggests that the SOVA decoder is less affected by the omission of some parity

information. It was also found that the rate of gain increase for both un-punctured and punctured codes decreased as the effective free distance of optimal component codes increased.

The effects of choosing optimal effective free distance were also apparent, codes with much smaller $d_{free,eff}$ that was optimal outperformed those with larger effective free distance that was not.

The choice of optimum decoder, when puncturing was used to reduce code rate, was related to signal-to-noise ratio. SOVA performed better for low signal-to-noise ratios under these circumstances for all codes. This was the same for frame error rate performance for un-punctured codes, however, the un-punctured bit error rate performance of the two decoders showed that Log-MAP was the better performer at all points along the curve.

Optimal effective free distance governed the order of performance for Log-MAP decoding, with smaller optimised component codes producing the best results at low signal-to-noise ratios and larger effective free distances producing better results at higher signal-to noise ratios. For the SOVA decoder, low signal-to-noise ratio performance followed the same route as Log-MAP, however, at high signal-to-noise ratios, lower optimal effective free distances were the better codes.

Larger datawords were then considered. The 512 bit datawords were still regarded as having relatively small frame length as generally turbo codes are investigated for frames with lengths over 1000 bits. These simulations only considered punctured turbo codes.

The longer frames produced improved results on those obtained previously, with SOVA benefiting more in terms gain from the increase than Log-MAP. The effects of incorrect component code choice were more apparent, the results of the 32-state code virtually matching those of the 4-state code. The choice of component code was again dependent on signal-to-noise ratio and the required level of performance. The rules were more concise for these longer frames, low optimal effective free distances performed better for low SNRs and high error rates, whereas high optimal effective free distances were best at high signal-to-noise ratios and low error rates.

In general, it was found that voice quality bit error rates were reached at lower SNRs for SOVA than they were for Log-MAP, but this did not continue for lower bit error rates at higher signal-to-noise ratios.

Error floors were more evident in these simulations, with drastic effects, particularly fir the Log-MAP decoders. This upheld the conclusions drawn from earlier simulations with regards to errors in the interleaver design methods. These methods did not consider the effects of puncturing on the turbo codeword and, as such, were highly unlikely to take into account the possibilities of some data bits having no parity information at all at the decoder. No error floor was found for the 16-state code, nor was one likely. This is most likely due to a fluke in interleaver design, whereby a very good version was obtained during the limited time that was available for design of the device using the chosen design system.

The effect of Rayleigh fading was then examined on punctured 100 bit dataword turbo codes. It was found that Rayleigh fading affected the decoders whether the fading information was available at the receiver or not. The gradients of the performance curves were altered, as well as the SNRs at which certain levels of performance were reached. These effects were found to be smaller for the Log-MAP

decoder than they were for the SOVA, suggesting that Log-MAP was more resilient under this type of distortion.

The effect of omitting fading information at the decoder did not catastrophically affect either decoder, but the performance was reduced and the gradients of the performance curves were further affected. This was more noticeable in the results obtained for the SOVA decoder, suggesting that this decoder is more reliant on fading information. This conclusion is borne out by the fact that SOVA only outperformed Log-MAP at low signal-to-noise ratios where fading information was available. The design of the turbo decoder with respect to the signal-to-noise ratio was again confirmed, with a smaller $d_{free,eff}$ proving the better performer at low SNRs and a larger $d_{free,eff}$ producing the better results at high SNRs.

Having established the fact that SOVA was less productive at high signal-to-noise ratios, but was generally similar to Log-Map at low ratios, and that Log-Map was less affected by fading effects or bad fading estimates at the decoder, the decision was made to investigate the complexity of the competing schemes.

It was found that, where the same component codes were considered, SOVA was vastly less complex than Log-MAP. This was shown through examination of published complexity estimates as well as software timing simulations. The two decoders were therefore simulated with comparable complexity. It was noted that this could be interpreted in two ways. First, the number of decoder iterations could be increased for the SOVA decoder and secondly the size of the component code could be increased for the SOVA decoder.

Initial comparisons were made for increased decoder iterations and it was found that SOVA outperformed Log-MAP for voice quality bit error rates with less or equal complexity, making this decoder algorithm a better choice for voice communications in terms of both complexity and performance. The Log-MAP decoders were still better at lower performance rates.

The two decoding algorithms were then briefly compared for the same number of decoder iterations and different component codes. It was found that under these circumstances, SOVA out performed Log-MAP at low signal-to-noise ratios, although this can hardly be considered conclusive evidence as there was a large error floor in the Log-Map results and only one comparison simulation was conducted.

It was also noted that there are in fact two pieces of channel information necessary for the turbo decoder to function properly. The first, fading amplitude was investigated earlier. The second piece of information was the signal-to-noise ratio of the transmission. No information was available on this subject for the SOVA decoder. Simulations were therefore undertaken to determine the effects of erroneous information at the SOVA turbo decoder. It was found that the SOVA algorithm reacted in a similar way to the Log-MAP algorithm, but was less affected in general. This was especially true where the size of the difference between the actual SNR and that utilised would ruin the performance of the Log-MAP decoder. At this point, the effect on the SOVA decoder remained negligible.

Through the course of these simulations, Log-MAP has been found to generally be the better performer at high signal-to-noise ratios for short datawords, in the same way as it was for long frame

turbo codes. It appears that SOVA codes of equivalent complexity may produce better results, but this has not been proved. What has also become apparent, however, is that the reasons for the ruling out of turbo codes for voice communications in recent times were largely due to assumptions brought on by the application of results for long frame turbo codes that were not applicable to short frame transmission.

Turbo codes have previously been omitted because they are assumed to be too slow. This may be the case for codes using Log-MAP, but the answer to this may be to consider the SOVA decoder instead. It is much less complex and therefore quicker, and with this reduction in complexity comes results equivalent to, or better than, its competitor.

It is true to say that SOVA is more susceptible to flaws in fading information at the decoder. The effect is more obvious than with Log-MAP, but is not catastrophic by any means. On the flipside, this decoder is less susceptible to errors in signal-to-noise ratio estimation at the receiver, proving to be unaffected at levels of error that caused failure in Log-MAP decoding. All this shows that, with careful attention paid to the construction of these codes, SOVA can equal Log-MAP with much less complexity at voice quality transmissions. It may also be true that for any given Log-MAP decoder, there is a SOVA decoder of equal complexity able to perform better at lower bit error rates.

Chapter

6

Equalisation and Combination with Turbo Codes

6.1 Introduction

As explained in chapter 5, error control coding reduces the effects of noise in the transmission channel. Mobile communications channels are not simply Additive White Gaussian Noise (AWGN) however. A transmitted signal can also be subject to fading effects, which is multiplicative rather than additive and is often modelled by the Rayleigh distribution. Another source of perturbation, often encountered on magnetic recording materials as well as mobile communications is Inter-Symbol Interference or ISI. The equaliser's role is to counteract the effects of this ISI.

To further understand the need for equalisation, it is perhaps pertinent to briefly reiterate some of the basic problems faced in digital communications channels. Figure 6.1 gives a very basic overview of the digital communications channel.

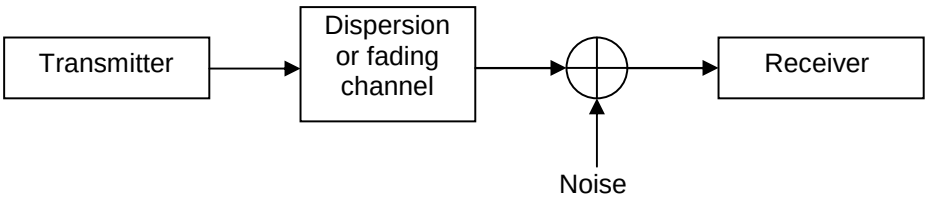


Figure 6.1: Digital Communications System Overview

Both channels, dispersion and fading, can cause Inter Symbol Interference (ISI). In the case of the dispersion channel, the continuous impulse response may spread over many symbol intervals,

therefore causing ISI. One of the properties associated with fading channels is the multipath effect, as demonstrated in figure 6.2.

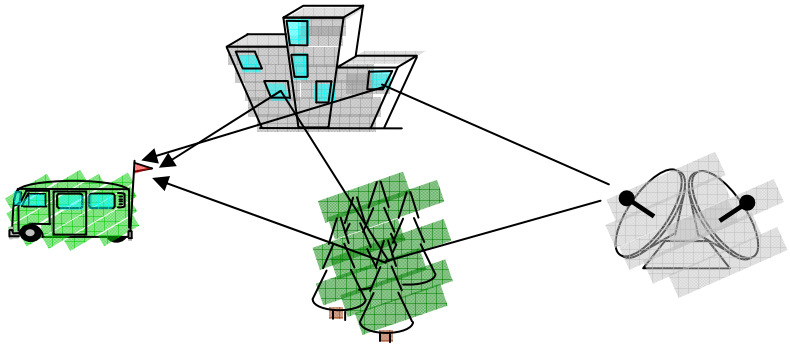


Figure 6.2: Multipath Propagation

The omni-directional nature of the transmitted signal implies that refraction and reflections can occur in the area between the transmitter and receiver. The distance travelled by each signal varies and they will therefore arrive at the receiver at different times. The delay spread is defined as the time between the arrival of the first and last signals. In reality the delay spread varies due to reflections from moving objects within the transmission medium.

If the delay spread is small when compared to the symbol period of the signal, the effect is referred to as flat fading. The received signal envelope will follow a Rayleigh or Rician distribution and the effects of ISI are negligible. If the delay spread exceeds the symbol period however, adjacent symbols become smeared into one another. This is frequency selective fading and can cause significant ISI, leading to an irreducible error floor.

Turbo codes have one important problem when it comes to decoding over fading channels, they require good channel information to effectively decode the received data. If channel information is not present, the decoder cannot scale the received information and will therefore not perform to its full potential.

The equaliser attempts to reconstruct the original transmitted information, removing the ISI, by observing the channel output $y(n)$.

In its most basic terms, if the transfer function of the channel is $H(z)$, then the transfer function of the ideal equaliser is clearly:

$$G(z) = H^{-1}(z) \tag{6.1}$$

This simple solution requires only the measurement of $H(z)$ and its inversion to produce a perfect reconstruction of the original signal. However, the mobile radio channel is under constant change and the reality is that the equaliser very rarely completely compensates for the effects of the channel. It is proposed that negating the fading effects of the channel should allow the turbo code to use a more general scaling factor, akin to that used for decoding over AWGN.

While the combination of Soft-Input Soft-Output (SISO) equalisation techniques in combination with one or more SISO decoders in a turbo-like structure (turbo equalisation) is currently receiving much attention, the combination of turbo codes with decision feedback equalisation remains relatively unexplored.

There is no doubt that the first combination produces excellent results, especially in situations exhibiting harsh Inter-Symbol Interference (ISI), but again, the complexity and delay of these systems can become an issue in systems that require quick and effective error control.

Decision feedback equalisers (DFEs) offer a much less complex solution to the reduction of ISI, having proved themselves time and again as adequate equalisers whilst keeping delay within acceptable limits. The complexity of these devices when compared with turbo decoding schemes is minimal, yet they produce very good results, curtailing the effects of even the harshest channels.

This chapter explores the implementation of the DFE and its combination with the turbo decoder strategy. Comparing the qualities of two of the most popular update algorithms and combining them with the turbo decoder strategies discussed earlier for short frame turbo codes.

The two update algorithms used here are the Least Mean Squares (LMS) algorithm and the Square Root Kalmann (SRK), a derivative of the Recursive Least Squares (RLS) algorithm. Both systems are based on the DFE architecture the derivations of which are also given. These were used to counteract the effects of four different channels prior to turbo decoding.

Of the four channels examined, one is static and time-invariant and the other three were time-varying, based on measured values [UMTS1] representing the effects of different situations, the indoor office environment, mobile pedestrian and mobile vehicular environments.

6.2 Adaptive Equalisation

As the purpose of an equaliser is to counteract the effects of ISI, the equaliser transfer function should be the inverse of that of the channel. For this, a simple tap delay line filter would suffice. However, the nature of the mobile communications channel is such that variations occur in signal reflections due to movement of the transmitter, receiver or of objects within the channel.

These constant variations in the effects of ISI, however small, can have an adverse effect on the performance of any coding scheme. This led to the development of adaptive equalisers. The techniques use basic learning algorithms to constantly update the transfer function of the equaliser in an effort to match the altering effects of the channel.

To accelerate the learning process, at regular intervals pilot symbols are transmitted. The pilot symbol sequence is information that is known to both the transmitter and receiver and can therefore be used as an aid in estimating the effects of the channel.

6.2.1 Linear Transversal Equalisers (LTE)

The Linear Transversal Equaliser (LTE), an example of which can be seen in figure 6.3, is a simple tap delay line equaliser. Both the peak distortion and the Mean Squared Error (MSE) criterion can be used to good effect when optimising the equaliser coefficients [GEO71] [PRO75] [MUE81] [SAL73].

Minimisation of the peak distortion assumes an equaliser with an infinite number of taps and a transfer function equal to the inverse of that of the channel, completely eliminating the ISI. Known as a zero-forcing transversal filter, the solution is found adaptively using a steepest descent recursive algorithm [LUC66] [PRO89].

The second method minimises the Mean Squared Error (MSE) between the desired outputs and those of the equaliser. The most common method, originally presented by Widrow and Hoff in 1960 [WID60] is the Least Mean Square (LMS) algorithm [GER69] [LUC68]. A stochastic gradient algorithm, the LMS iterates each tap weight of the transversal filter, following the gradient of the squared amplitude of the error signal with respect to the tap weight. The algorithm is similar to stochastic approximation methods, however, the step size that controls the correction to each tap weight remains constant in the LMS, whereas the step size varies with time in stochastic approximation. This algorithm is the most commonly used to reduce MSE due its low complexity and robustness.

Linear Equalisers have limited application in mobile communications as the channels themselves are rarely linear and on channels with deep spectral nulls the noise can be enhanced, hence the poor performance when frequency selective fading is encountered. This realisation led to the development of adaptive non-linear equalisation techniques of which the Maximum Likelihood Sequence Estimator (MLSE) [FOR72] is the best available.

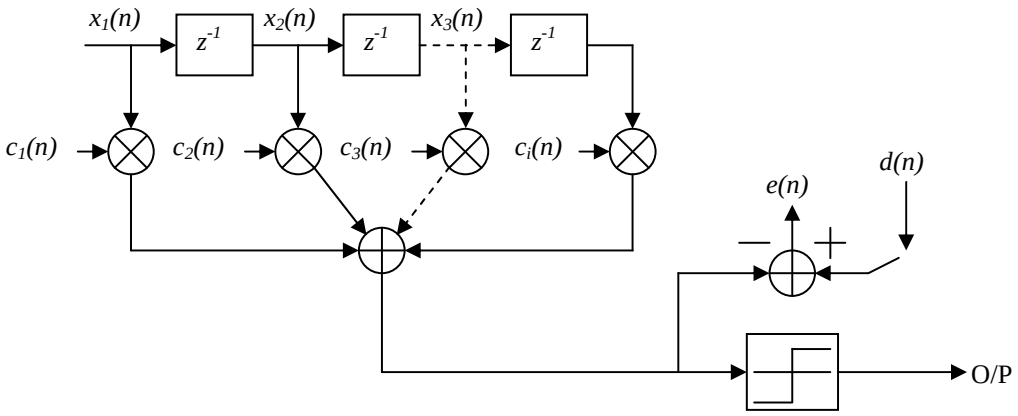


Figure 6.3: Example of Linear Transversal Equaliser (LTE)

$\boldsymbol{x}^t(n)$ is the input vector at time instant n and is defined as:

$$\mathbf{x}^t(n) = \left[x_1(n), x_2(n), ..., x_N(n)\right] \tag{6.2}$$

$\boldsymbol{c}^t(n)$ are the equaliser coefficients at time instant n, defined as:

$$\mathbf{c}^t(n) = \left[c_1(n), c_2(n), ..., c_N(n)\right] \tag{6.3}$$

$d(n)$ is the transmitted signal during training mode and the output of decision device during the decision direct mode, once training is complete.

$$y(n) = \mathbf{x}^t(n)\mathbf{c}(n-1) \tag{6.4}$$

And is the output of the equaliser before the decision device.

The error signal is defined as:

$$e(n) = d(n) - y(n) = d(n) - \mathbf{x}^t(n)\mathbf{c}(n-1) \tag{6.5}$$

If the channel response is unknown or time varying then $\mathbf{c}(n)$, the vector defining the coefficients of the equaliser needs to be determined adaptively.

Note that in equations defining $y(n)$ and $e(n)$, the vector from the previous time instant is used because the filter is adaptive and therefore the best coefficients available at instant n are those generated immediately before.

6.2.2 Decision Feedback Equalisers (DFE)

The Decision Feedback Equaliser (DFE) [GEO71] [SAL73], see figure 6.4, is a simple non-linear equaliser, comprised of a feedforward filter and a feedback filter, both of which have taps spaced at the symbol interval. The feedforward filter is an LTE, while the feedback filter takes as its input a series of previous decisions. The theory being that the feedback filter removes ISI attributable to these symbols by subtracting properly weighted versions from the equaliser output. Of course, this works only if the decisions were correct.

Whereas the coefficients of an LTE are selected such that the impulse response of the channel and equaliser approximates a unit pulse, with the DFE, the fact that the feedback filter is used to eliminate the ISI means that there are fewer restrictions when choosing the coefficients of the feedforward filter. There is an obvious problem with this system, which occurs if an incorrect decision is returned through the feedback filter. As a weighted version of the incorrect decision is subtracted

from future symbols output from the equaliser, the chances of other incorrect decisions being made, and fed back into the system, is increased, further degrading the performance. Fortunately, this effect will not continue indefinitely, but tends to produce bursts of errors. In [HAY01], the author gives the following intuitive reasoning for this, following [GIT92]. After a sequence of correct decisions, equal to the number of taps in the feedback filter (A), any errors in the feedback section will be flushed out, indicating that the error propagation is limited. Also, the likelihood that the decision following an incorrect decision will itself be incorrect is no more than 1/2. If B is the duration of error propagation (i.e. the number of received symbols required to make A correct decisions and therefore flush out an error in the feedback system), then the average number of errors produced by a single decision is equal to $B/2$ and the average error rate is $B/2$ multiplied by the probability of error given that the last A decisions have been correct. In a fair-coin tossing experiment (where heads represent correct decisions and tails represent incorrect decisions), the average number of tosses, B, required to get A successive heads is $2(2^A-1)$. The effect of error propagation is therefore to increase the average error rate by a factor of approximately 2^A. Despite this fact, the DFE is a simple system, much less complex than the Maximum Likelihood Sequence Estimator (MLSE), with only a small reduction in performance, making it a common choice when combating ISI.

The feed forward filter of a DFE is an LTE, sampled at the symbol rate, and therefore very sensitive to sampling time. To avoid any performance reduction, the input to the equaliser is sampled at a rate higher than once per symbol.

The LMS algorithm has a long convergence time, particularly when the propagation channel is highly time variant as in mobile communications. To better track rapidly fluctuating channel state variation and therefore improve convergence time, the Recursive Least Squares (RLS) algorithm, or a derivation thereof, can be used. Closely related to the Kalman filter, the RLS algorithm is sensitive to round-off errors and can become unstable. Variations of this algorithm, for instance, the Square Root Kalman (SRK) algorithm, are less numerically precise but possess better stability.

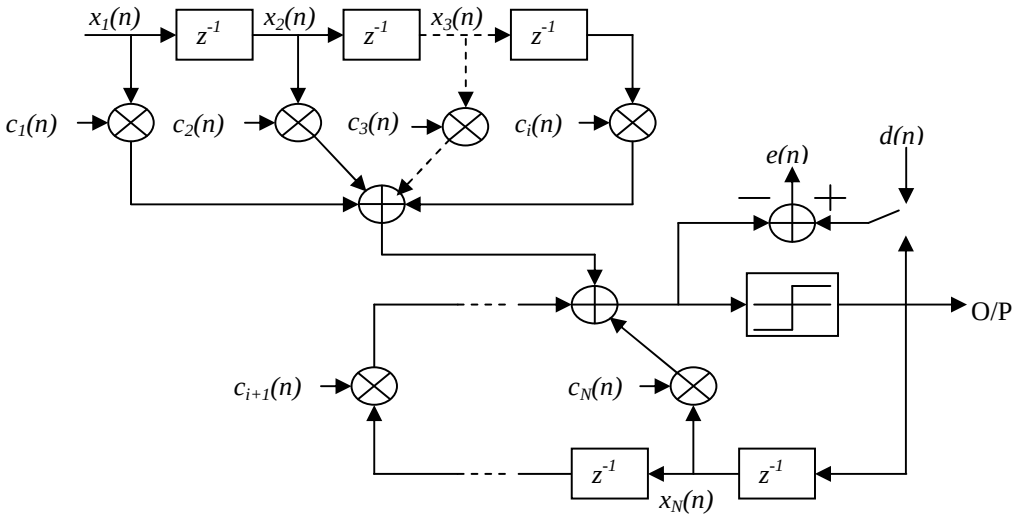


Figure 6.4: Example of Decision Feedback Equaliser (DFE)

6.2.3 Least Mean Square (LMS) Algorithm

The LMS algorithm is the simplest method for updating a decision feedback type equaliser, this is due to the fact that the algorithm does not require the measurement of pertinent correlation functions or matrix inversion.

$\nabla(J(n))$ is the value of the gradient vector at time *n*

J(n) is the mean square error

$$J(n) = E\left\{|e(n)|^2\right\} \tag{6.6}$$

Where *E* indicates the expected, or average, value.

According to the method of steepest descent [HAY91], the updated value of the tap-weight vector at time *n* is:

$$\mathbf{c}(n) = \mathbf{c}(n-1) + \frac{1}{2}\mu\left[-\nabla(J(n))\right] \tag{6.7}$$

Where μ is a positive and real-valued constant called the Adaptive Constant, and the factor ½ is used for convenience.

$$\nabla\left(J(n)\right)=\frac{\partial J(n)}{\partial \mathbf{c}(n-1)}=\begin{bmatrix} \frac{\partial J(n)}{\partial Re\left[c_1(n-1)\right]}+j\frac{\partial J(n)}{\partial Im\left[c_1(n-1)\right]} \\ \frac{\partial J(n)}{\partial Re\left[c_2(n-1)\right]}+j\frac{\partial J(n)}{\partial Im\left[c_2(n-1)\right]} \\ \vdots \\ \frac{\partial J(n)}{\partial Re\left[c_N(n-1)\right]}+j\frac{\partial J(n)}{\partial Im\left[c_N(n-1)\right]} \end{bmatrix} \tag{6.8}$$

Instead of the true gradient of mean square error, $\nabla(J(n))$, the LMS algorithm uses the instantaneous estimate $\hat{\nabla}(J(n))$:

$$\hat{\nabla}(J(n))=\frac{\partial |e(n)|^2}{\partial \hat{\mathbf{c}}(n-1)}=-2e(n)\mathbf{x}^*(n) \tag{6.9}$$

Substituting this estimate for the true gradient in equation (6.8) produces the following coefficient update estimates:

$$\hat{\mathbf{c}}(n)=\hat{\mathbf{c}}(n-1)+\mu e(n)\mathbf{x}^*(n) \tag{6.10}$$

Where * denotes complex conjugation.

For the LMS algorithm, the mean error converges to zero as n approaches infinity as long as ([UNG72], [WID70])

$$0<\mu<\frac{2}{\lambda_{\max}} \tag{6.11}$$

Where λ_{max} is the largest eigen value of the correlation matrix of the input vector. Under this condition, the LMS is referred to as 'convergent in the mean'.

6.2.4 Recursive Least Squares (RLS) Algorithm

A more complex update algorithm for use with the DFE is the RLS. This algorithm can take into account information contained in the input data from the earliest received information up to the most recent, meaning that the rate of convergence tends to be faster than that of the LMS algorithm. The complexity of computation, however, is increased.

The cost function to be minimised is expressed as $\varepsilon(n)$ where n is the variable length of the observable data.

The RLS algorithm also introduces a 'weighting' factor, which effectively assigns more importance to recent data. One method of accomplishing this is to define $\varepsilon(n)$ as an exponentially weighted sum of squares:

$$\varepsilon(n) = \sum_{i=1}^{n} \omega^{n-i} |e(i)|^2 \tag{6.12}$$

Where $e(i)$ is as defined in equation (6.5) and ω is a constant factor close to, but less than one, which determines the effective memory of the algorithm. This 'weighting' factor is particularly useful when operating in a non-stationary environment, allowing old data to be forgotten and therefore allowing the algorithm to follow the statistical variations in the observable data.

Effective memory with this algorithm is approximately $1/(1 - \omega)$ data points, thus $\omega = 1$ implies that the algorithm would have infinite memory.

The optimum value of the equaliser coefficients vector, $\hat{\mathbf{c}}(n)$, which minimises $\varepsilon(n)$ is defined by the following equations, written in matrix form:

$$\Phi(n)\hat{\mathbf{c}}(n) = \theta(n) \tag{6.13}$$

The *N by N* correlation matrix $\Phi(n)$ is defined as:

$$\Phi(n) = \sum_{i=1}^{n} \omega^{n-i} \mathbf{x}^*(i)\mathbf{x}^t(i) \tag{6.14}$$

The *N by 1* cross-correlation vector, $\theta(n)$ between the tap inputs of the transversal filter and the desired response is defined as:

$$\theta(n) = \sum_{i=1}^{n} \omega^{n-i} \mathbf{x}^*(i)d(i) \tag{6.15}$$

Where * denotes the complex conjugate and ^{t} is matrix transposition. Note also that pre-windowing is assumed, that is that the input data before $i = 1$ is taken to be zero.

The recursive update of the tap inputs for $\Phi(n)$, the correlation matrix can be expressed as:

$$\Phi(n) = \omega\left[\sum_{i=1}^{n} \omega^{n-1-i} \mathbf{x}^*(i)\mathbf{x}^t(i)\right] + \mathbf{x}^*(n)\mathbf{x}^t(n) = \omega\Phi(n-1) + \mathbf{x}^*(n)\mathbf{x}^t(n) \tag{6.16}$$

Where $\Phi(n-1)$ is the previous value of the correlation matrix and by definition is the value in the square parentheses. Here, $\mathbf{x}^*(n)\mathbf{x}^t(n)$ is a correction term in the updating operation.

In the same way, this recursive process can be applied to updating the cross-correlation vector between the tap inputs and the desired response $\theta(n)$:

$$\theta(n) = \omega\left[\sum_{i=1}^{n} \omega^{n-1-i}\mathbf{x}^*(i)d(i)\right] + \mathbf{x}^*(n)d(n) = \omega\theta(n-1) + \mathbf{x}^*(n)d(n) \tag{6.17}$$

To compute the least square estimate for the vector $\mathbf{c}(n)$ as stated in equation (6.13), the inverse of the correlation matrix $\Phi(n)$ must first be determined. However, if the number of tap-weights (N) is high, this can be time consuming. To avoid this, the Matrix Inversion Lemma (MIL) (A full derivation of which can be found in Appendix L), a basic matrix algebra function, is used.

Considering two positive-definite *N by N* matrices, *A* and *B*, related by:

$$A = B^{-1} + CD^{-1}C^H \tag{6.18}$$

Where *D* is another positive-definite matrix, this time with dimensions *M by M*. *C* is an *N by M* matrix and ^{H} represents Hermitian transposition (transposition and complex conjugation combined).

Matrix Inversion Lemma states that A^{-1} can be expressed as:

$$A^{-1} = B - BC(D + C^H BC)^{-1}C^H B \tag{6.19}$$

The MIL can now be used to obtain a recursive equation to compute a solution to the least squares for the coefficient vector $\mathbf{c}(n)$.

Assuming that the correlation matrix $\Phi(n)$ is positive-definite and therefore non-singular, the MIL can be applied using the following:

$$A = \Phi(n)$$
$$B^{-1} = \omega\Phi(n-1)$$
$$C = \mathbf{x}^*(n)$$
$$D = 1$$

Substituting these into the MIL gives the recursive equation for the inverse of the correlation matrix:

$$A^{-1} = B - BC(D + C^H BC)^{-1}C^H B$$

$$\Phi^{-1}(n) = \omega^{-1}\Phi^{-1}(n-1) - \left(\frac{1}{1+\mathbf{x}^t(n)\omega^{-1}\Phi^{-1}(n-1)\mathbf{x}^*(n)} \right) \omega^{-1}\Phi^{-1}(n-1)\mathbf{x}^*(n)\mathbf{x}^t(n)\omega^{-1}\Phi^{-1}(n-1) \tag{6.20}$$

$$\Phi^{-1}(n) = \omega^{-1}\Phi^{-1}(n-1) - \left(\frac{\omega^{-2}\Phi^{-1}(n-1)\mathbf{x}^*(n)\mathbf{x}^t(n)\Phi^{-1}(n-1)}{1+\mathbf{x}^t(n)\omega^{-1}\Phi^{-1}(n-1)\mathbf{x}^*(n)} \right) \tag{6.21}$$

Making the following definitions:

$$P(n) = \Phi^{-1}(n) \tag{6.22}$$

$$k(n) = \frac{\omega^{-1}P(n-1)\mathbf{x}^*(n)}{1+\omega^{-1}\mathbf{x}^t(n)P(n-1)\mathbf{x}^*(n)} \tag{6.23}$$

Means that equation (6.21) can be re-written as:

$$P(n) = \omega^{-1}P(n-1) - \omega^{-1}k(n)\mathbf{x}^t(n)P(n-1) \tag{6.24}$$

$$P(n) = \omega^{-1}\left[P(n-1) - k(n)\mathbf{x}^t(n)P(n-1) \right] \tag{6.25}$$

P(n) is an *N by N* matrix while *k(n)* is a vector of dimensions *N by 1* and is often referred to as the gain vector. Equation (6.25) is called the Riccati equation for the RLS algorithm.

Rearranging (6.23) gives:

$$k(n) + k(n)\omega^{-1}\mathbf{x}^t(n)P(n-1)\mathbf{x}^*(n) = \omega^{-1}P(n-1)\mathbf{x}^*(n) \tag{6.26}$$

$$k(n) = \omega^{-1}P(n-1)\mathbf{x}^*(n) - k(n)\omega^{-1}\mathbf{x}^t(n)P(n-1)\mathbf{x}^*(n) \tag{6.27}$$

$$k(n) = \omega^{-1}\left[P(n-1)\mathbf{x}^*(n) - k(n)\mathbf{x}^t(n)P(n-1)\mathbf{x}^*(n) \right] \tag{6.28}$$

$$k(n) = \omega^{-1}\left[P(n-1) - k(n)\mathbf{x}^t(n)P(n-1) \right]\mathbf{x}^*(n) \tag{6.29}$$

$$k(n) = P(n)\mathbf{x}^*(n) \tag{6.30}$$

$P(n) = \Phi^{-1}(n)$, therefore the gain vector can be defined as the conjugate of the input vector transformed by the inverse of the correlation matrix.

$$k(n) = \Phi^{-1}(n)\mathbf{x}^*(n) \tag{6.31}$$

Therefore, using equation (6.13), the least-squared estimated coefficient vector can be expressed as:

$$\hat{\mathbf{c}}(n) = \Phi^{-1}(n)\theta(n) \tag{6.32}$$

$$\hat{\mathbf{c}}(n) = P(n)\theta(n) \tag{6.33}$$

And using equation (6.17)

$$\hat{\mathbf{c}}(n) = P(n)\left[\omega\theta(n-1) + \mathbf{x}^*(n)d(n)\right] \tag{6.34}$$

$$\hat{\mathbf{c}}(n) = \omega P(n)\theta(n-1) + P(n)\mathbf{x}^*(n)d(n) \tag{6.35}$$

Substituting (6.25) for $P(n)$ into equation (6.35) gives:

$$\hat{\mathbf{c}}(n) = \omega\omega^{-1}\left[P(n-1) - k(n)\mathbf{x}^t(n)P(n-1)\right]\theta(n-1) + P(n)\mathbf{x}^*(n)d(n) \tag{6.36}$$

$$\hat{\mathbf{c}}(n) = P(n-1)\theta(n-1) - k(n)\mathbf{x}^t(n)P(n-1)\theta(n-1) + P(n)\mathbf{x}^*(n)d(n) \tag{6.37}$$

$$\hat{\mathbf{c}}(n) = \Phi^{-1}(n-1)\theta(n-1) - k(n)\mathbf{x}^t(n)\Phi^{-1}(n-1)\theta(n-1) + P(n)\mathbf{x}^*(n)d(n) \tag{6.38}$$

$$\hat{\mathbf{c}}(n) = \hat{\mathbf{c}}(n-1) - k(n)\mathbf{x}^t(n)\hat{\mathbf{c}}(n-1) + P(n)\mathbf{x}^*(n)d(n) \tag{6.39}$$

Substituting equation (6.30) gives:

$$\hat{\mathbf{c}}(n) = \hat{\mathbf{c}}(n-1) - k(n)\mathbf{x}^t(n)\hat{\mathbf{c}}(n-1) + k(n)d(n) \tag{6.40}$$

$$\hat{\mathbf{c}}(n) = \hat{\mathbf{c}}(n-1) + k(n)\left[d(n) - \mathbf{x}^t(n)\hat{\mathbf{c}}(n-1)\right] \tag{6.41}$$

Considering equation (6.5):

$$\hat{\mathbf{c}}(n) = \hat{\mathbf{c}}(n-1) + k(n)e(n) \tag{6.42}$$

Here *e(n)* can be called the *a priori* estimation error as $\mathbf{x}^t(n)\hat{\mathbf{c}}(n-1)$ is an estimation of the desired response *d(n)* based upon the least squares estimate of the coefficient vector made at time *(n – 1)*.

The RLS algorithm requires Soft Constrained Initialisation. That is to say that the values of matrix *P(0)* must be initialised as:

$$P(0) = \delta^{-1} I \tag{6.43}$$

Where *I* is the *N by N* identity matrix and δ is a small positive constant, thus ensuring the nonsingularity of the correlation matrix $\Phi(n)$, and that the initial value for the coefficient vector is initialised as:

$$\hat{\mathbf{c}}(0) = 0 \tag{6.44}$$

Where, *0* represents the *N by 1* null vector.

Hubing and Alexander [HUB90] have shown, through statistical analysis of Soft Constrained Initialisation, that δ should be small when compared with $0.01\sigma^2_x$, where σ^2_x is the variance of the data sample *x(n)*. While Haykin [HAY91] states that this is unimportant for large data lengths.

6.3 Combined Equaliser with Turbo Codes System Model

The system employed follows conventional equaliser and error control coding combinations. Figure 6.5 below helps to illustrate this.

Applying turbo codes to the system above, once data has been encoded it is punctured, channel interleaved (using a block interleaver) and modulated using BPSK. The transmitted codeword is then subjected to the effects of ISI and AWGN. At the receiver, the corrupted signal is demodulated and channel de-interleaved prior to equalisation. The equaliser attempts to counteract the ISI before the complete codeword can be passed to the turbo decoding system.

Figure 6.6 below describes the DFE in combination with the turbo decoder. For the sake of simplicity, the turbo decoder is shown as a single block.

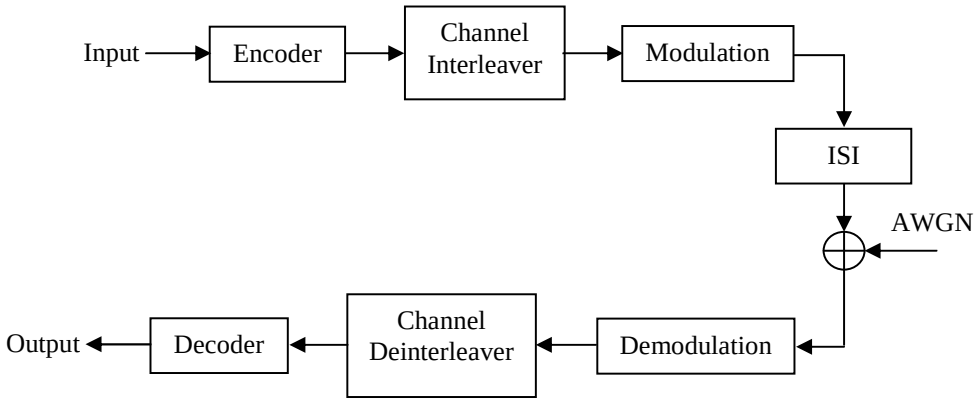


Figure 6.5: System Combining Equaliser with Channel Coding and Block Type Channel Interleaver

The red line in the figure below indicates the simple modification used to combine the equaliser with the turbo decoder. In conventional decoding techniques, the hard output of the equaliser decision device is used as the input to the decoder. Turbo decoders require a soft input to produce their excellent results and therefore the hard input should have a detrimental effect on the quality of the outputs of the two decoding algorithms.

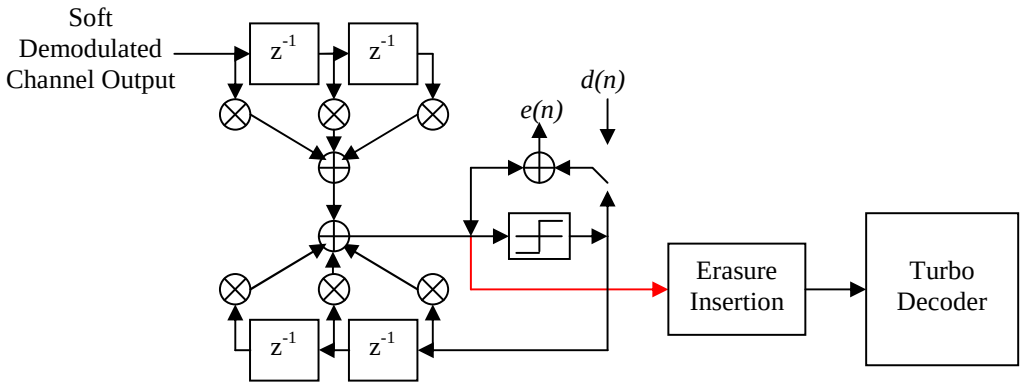


Figure 6.6: Combining DFE with Turbo Decoder
(Erasure insertion replaces punctured bits with null values)

Although the values obtained before the decision device of the equaliser are not soft in the proper sense, it was found that the decoder responded more favourably to these values, therefore the input to the decision device is used, prior to erasure insertion, serial to parallel conversion and ultimately decoding. The fact remained, however, that once the equaliser had stabilised (in the case of the time-invariant channel at least), the input to the decision device was virtually binary.

6.4 Implementing Decision Feedback Equalisers

The program allows for different sizes of feed forward and feedback filters as well as the adaptive constant μ for LMS and ω, the weighting or forgetting factor for SRK.

DFEs require the insertion of pilot symbols within the transmitted frames. These pilot symbols and their positions in the transmitted frames are known at the receiver and are used as an aid to estimating the effects of the channel on the data. For a static channel, the frequency of these pilot symbols is dependent upon how badly the channel distorts the transmitted data.

Both equalisers take as their input demodulated symbols, prior to making hard decisions.

When implementing the DFE in simulation software, the delay effects must be taken into account before any information is passed on to the decoder. Symbol delays can occur at the simulated channel when working with tap-delay line models and at the equaliser itself. Both delays are determined by the position of the strongest tap of the particular tap delay model, in the case of the decision feedback equaliser, this is with respect to the forward filter. For the channel models used, the first tap has the most powerful multiplicative element therefore the channel delay is zero. The strongest tap for both the LMS and SRK equalisers is to be found in the centre position of the forward filter. These delays affect the point at which teaching begins. Due to the forward filter delay, it is also necessary that the equalisation of the subsequent frame begin before the current frame can be passed to the decoding structure.

For both equalisers certain initialisations are necessary, these are explained in the sections of this chapter pertaining to each specific algorithm.

Once initialisations have taken place, the frame can be processed. The forward filter of the equaliser takes as its input the next demodulated channel output, *soft_symbols*. This is used to calculate the output of the filter, *dfe_v*. The feedback filter takes as its input the current equaliser hard decision, *dfe_y*, and uses it to calculate the filter output *dfe_z*. The sum of these outputs forms the input to the hard decision device.

In the code below, *bcoeff* and *fcoeff* are vectors containing the coefficients of the backward and forward filters respectively and are initialised to zero.

Once the output of the two filters has been determined, the sum of those outputs is passed to the decision device. This sum is also stored to be used by the turbo decoding system as the soft input in the same way that the soft demodulated values were prior to using the equaliser.

```
for eqcount = length(soft_symbols) to 1 in steps of –1

      for shift = fwd_store to 1 in steps of –1
            %shift demodulated symbol into forward filter
            if shift > 1
                  fwd_store(shift) = fwd_store(shift-1)
            else
                  fwd_store(shift) = soft_symbols(eqcount)
            end
      end

      %calculate filter output
      dfe_v = 0
      for count1 = 1 to length(fwdcoeff)
            dfe_v = dfe_v + (fwd_store(count1)*fcoeff(count1))
      end

      for shift = bwd_store to 1 in steps of –1
            %shift eqout into backward filter
            if shift > 1
                  bwd_store(shift) = bwd_store(shift-1)
            else
                  bwd_store(shift) = eqout
            end
      end

      calculate filter output
      dfe_z = 0
      for count1 = 1 to length(bwdcoeffs)
            dfe_z = dfe_z + (bwd_store(count1)*bcoeff(count1))
      end
```

DFE 1: Calculating the Forward and Backward Filter Outputs

```
dfe_w = dfe_v + dfe_z

soft_eq(opcount) = dfe_w

%decision device
if dfe_w < = 0
      dfe_y = -1
else
      dfe_y = 1
end
```

DFE 2: Equaliser Decision Device

Once the hard decision has been made, the equaliser calculates the error between the hard decision and the soft decision. If the equaliser is in training mode, the hard decision is already known. When equalisation is taking place on the transmitted data, the error is calculated between the hard decision device output and the soft input *dfe_w*.

```
if teachcount <= totaldelay
%delay of channel and equaliser fwd filter
      error = 0 – dfe_w
else if remainder of (teachcount/teach)
%teach determines pilot symbol spacing
      error = pilot(teacher) – dfe_w
      teacher = teacher + 1
else
      error = dfe_y – dfe_w
end
```

DFE 3: Error Calculation

The counter *teacher* is initialised to one every time a frame is received and is used to determine the points at which the error signal calculations switch from learning mode to decision-direct mode, pilot is an array of pilot symbols.

The update algorithm acts on this error value, altering the coefficients associated with both the feed forward and feedback filters in an effort to adapt to the channel characteristics.

6.4.1 Implementing LMS

Before the equaliser can begin its process, certain initialisations must be made. Both of the equaliser filter coefficient vectors are set to zero, while the total delay (sum of channel delay and equaliser delay) and the step size parameter μ are manually specified.

The LMS is the simpler of the two equaliser algorithms described. μ controls the change in the filter coefficients, hence it is sometimes referred to as the step size parameter.

```
for lmscount =1 to length(fcoeff)
      fcoeff(lmscount) = fcoeff(lmscount) +
                              (mu*error*fwd_store(lmscount))
end

for lmscount = 1 to length(bcoeff)
      bcoeff(lmscount) = bcoeff(lmscount) +
                              (mu*error*fwd_store(lmscount))
end
```

LMS 1: Updating Filter Coefficients using the LMS algorithm

6.4.2 Implementing SRK

A derivative of the RLS algorithm, SRK is designed to avoid the numerical instability caused by the basic RLS update of the inverse of the correlation matrix *P(n)* The RLS process does not guarantee positive definiteness and therefore, the SRK algorithm utilises U-D factorisation, allowing *P(n)* to be represented as:

$$P(n) = U^{*}(n)D(n)U^{t}(n) \tag{6.45}$$

Where *U(n)* is an upper triangular matrix and *D(n)* is a diagonal matrix with real, positive elements. Updates are made on *U(n)* and *D(n)* rather than directly on *P(n)*. The initialisation of these elements is described below

```
%initialisations for forward filter
for setup = 1 to length(fcoeff)
      pf(setup) = 1                 %diagonal elements of D(0)
end

for setup = 1 to (length(fcoeff) – 1)
      for index = (setup + 1) to length(fcoeffs)
            uf(setup,index)=0 %upper triangular portion of U(0)
      end
end

%initialisations for backward filter
for setup = 1 to length(bcoeff)
      pb(setup) = 1
end

for setup = 1 to (length(bcoeff) – 1)
      for index = (setup + 1) to length(bcoeffs)
            ub(setup, index) = 0
      end
end
```

SRK 1: Initialising the Forward and Backward Filter Updates

All other elements require initialisation in much the same way as the LMS. There is no μ value, instead a forgetting factor, ω, which must be specified. ω controls the effective memory of the algorithm. The strength of the RLS algorithm (the SRK derivative does not alter the performance of the algorithm, it is merely an adaptation that avoids numerical instability) is that it takes into account previous information when updating the filter coefficients.

The update procedure for this algorithm is more complex than LMS and can be explained thus:

```
for srkcount = 1 to length(fcoeff)
      if srkcount == 1
            ff(srkcount) = fwd_store(srkcount)
            kf(srkcount) = pf(srkcount)*ff(srkcount)
            alphaf(srkcount) = omega+(kf(srkcount)*
                                        ff(srkcount))
      else
            sum = 0
            for count1 = 1 to (srkcount – 1)
                  sum = sum + (uf(count1, srkcount)*
                                        fwd_store(count1))
            end
      ff(srkcount) = sum + fwd_store(srkcount)
      kf(srkcount) = pf(srkcount)*ff(srkcount)
      alphaf(srkcount) = alphaf(srkcount – 1) +
                              (kf(srkcount)*ff(srkcount))
      end
end

kflast = kf
```

SRK 2: Beginning the Forward Filter Update

Having calculated *ff*, *kf* and *alphaf* for the forward filter, *pf* and *uf* are calculated prior to updating *kf* and finally determining epsilon, with which the forward filter coefficients will be updated.

```
      for srkcount = 1 to length(fcoeff)
            if srkcount == 1
                  pf(srkcount) = pf(srkcount)/alphaf(srkcount)
            else
                  pf(srkcount) = pf(srkcount)*alphaf(srkcount-1)/
                                          (omega*alphaf(srkcount))
                  for count1 = 1 to (srkcount – 1)
                        olduf = uf(count1, srkcount)
                        uf(count1, srkcount) =uf(count1, srkcount)
                  -(kf(count1)*(ff(count1)/alphaf(srkcount – 1))
                        kf(count1) = kflast(count1)+
                                          (kflast(srkcount)*olduf)
                  end
            end
      end
```

SRK 3: Continuing the Update

Once the vector *kf* has been updated, the forward filter coefficients can be determined.

```
      epsilon = error*alphaf(length(fcoeff))
      for srkcount = 1 to length(fcoeff)
            fcoeff(srkcount) = fcoeff(srkcount) + (kf(srkcount)*
                                                epsilon))
      end
end
```

SRK 4: Updating the Forward Filter Coefficients

Before the next symbol can be shifted into the equaliser's forward filter, the same update procedure must be applied to the backward filter.

```
      for srkcount = 1 to length(bcoeff)
            if srkcount == 1
                  fb(srkcount) = bwd_store(srkcount)
                  kb(srkcount) = pb(srkcount)*fb(srkcount)
                  alphab(srkcount) = omega+(kb(srkcount)*
                                          fb(srkcount))
            else
                  sum = 0
                  for count1 = 1 to (srkcount – 1)
                        sum = sum + (ub(count1, srkcount)*
                                          bwd_store(count1))
                  end
            fb(srkcount) = sum + bwd_store(srkcount)
            kb(srkcount) = pb(srkcount)*fb(srkcount)
            alphab(srkcount) = alphab(srkcount – 1) +
                                    (kb(srkcount)*fb(srkcount))
            end
      end
      kblast = kb
      for srkcount = 1 to length(bcoeff)
            if srkcount == 1
                  pb(srkcount) = pb(srkcount)/alphab(srkcount)
            else
                  pb(srkcount) = pb(srkcount)*alphab(srkcount-1)/
                                          (omega*alphab(srkcount))
                  for count1 = 1 to (srkcount – 1)
                        oldub = ub(count1, srkcount)
                        ub(count1, srkcount) =ub(count1, srkcount)
                  -(kb(count1)*(fb(count1)/alphab(srkcount – 1))
                        kb(count1) = kblast(count1)+
                                          (kblast(srkcount)*oldub)
                  end
            end
      end
      epsilon = error*alphab(length(bcoeff))
      for srkcount = 1 to length(bcoeff)
            bcoeff(srkcount) = bcoeff(srkcount) + (kb(srkcount)*
                                                epsilon))
      end
end
```

6.5 Combining DFEs with Turbo Codes

Before combining the two major components it was necessary to determine the specifications for the equalisers to be used to combat the effects of the channel.

Both the SRK and LMS algorithms required experimentation to determine the number of filter coefficients, while it was necessary to determine μ for the LMS algorithm and ω for the SRK algorithm. Figures 6.7 and 6.8 below show the effects that altering the memory coefficient (ω) and step size parameter (μ) have on the bit error rate of the equaliser hard output.

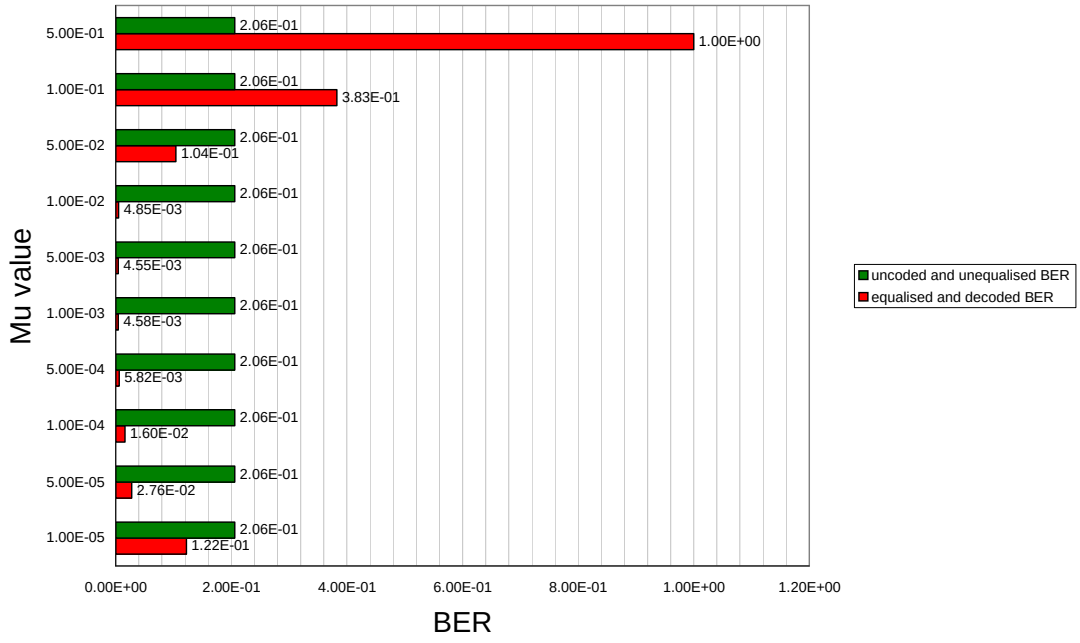


Figure 6.7: The Effects of Changing the Step Size Parameter (μ) with LMS Equalisation for the Time-Invariant Channel

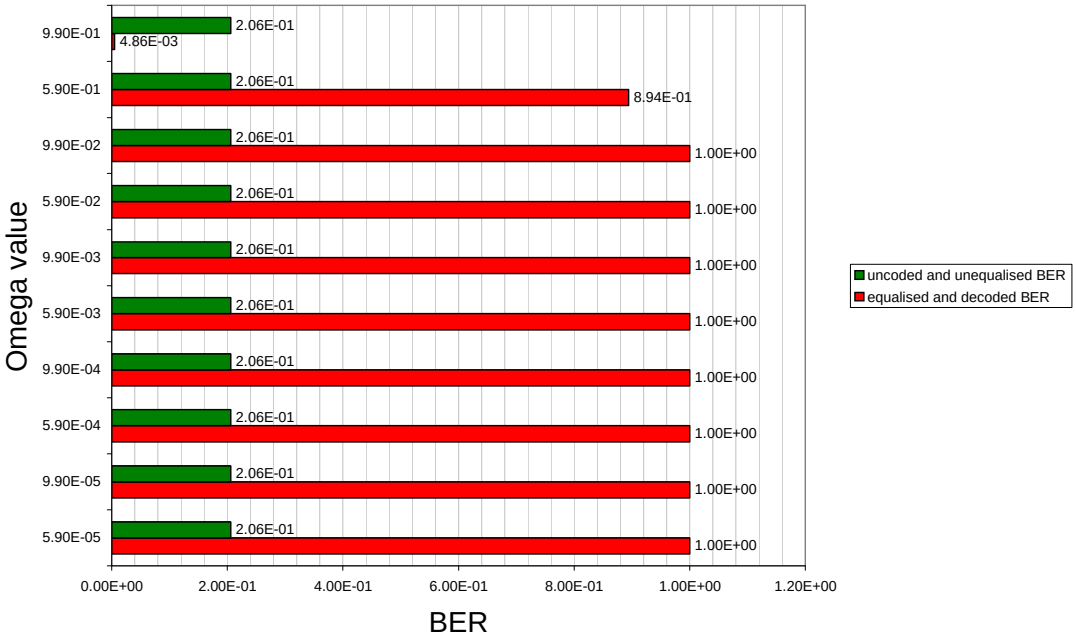


Figure 6.8: The Effects of Changing the Memory Coefficient (ω) with SRK Equalisation for the Time-Invariant Channel

Through experimentation and from the two figures above, the chosen values for the step size parameter (μ) of the LMS and the memory coefficient (ω) of the SRK equaliser were $5e^{-3}$ and $9.9e^{-1}$ respectively. Once the optimum equaliser had been determined for each update algorithm, the next stage was combining the two components, equaliser and turbo decoder.

6.6 Decoder and Equaliser Combined Results

As mentioned in section 6.2.2, the DFE is a combination of two filters. The feed forward filter output is the sum of a number of weighted received symbols while the feed back filter output is the sum of a number of weighted previous hard decisions. The system presented here feeds the sum of these two outputs to the turbo decoder. In doing so, the soft value of the original received symbol is compromised because the efforts to remove the ISI have also altered the additive noise effects. Knowledge of the noisy received symbol provides the component decoders with extra information that can be passed between them to improve the BER, iteration to iteration. The correct method of scaling the data received from the equaliser prior to decoding is therefore of great significance.

As to the correct method of scaling this data, two scenarios were considered. If the equaliser had performed properly, the effect of the channel on the transmitted data would be reduced to that of AWGN. Therefore, to begin with, AWGN channel fading values were used at the decoder, meaning that the equaliser output would be subjected to the same scaling factor that would be applied to turbo codes transmitted over this type of channel. It is also possible to argue that the fact that the equaliser

has pre-processed the transmitted information could mean that all channel information has been lost within the received data. The first group of simulations were therefore followed by an equivalent set with the channel reliability completely omitted (equivalent to using a channel reliability of $L_c = 1$ at all SNRs).

6.6.1 Time-Invariant Simulations with AWGN Reliability

The LMS and SRK update algorithms were applied, in combination with SOVA and Log-MAP turbo decoders, over the channels described in chapter 3. Ideal signal-to-noise information was assumed and two component codes were simulated, the $[7;5]_8$ four state code and the $[23;33]_8$ 16 state code, with interleaver lengths of 512. Codewords were punctured according to the method described in section 4.4.5 and the interleavers used were those obtained earlier in the thesis using the method described in section 4.4.4.

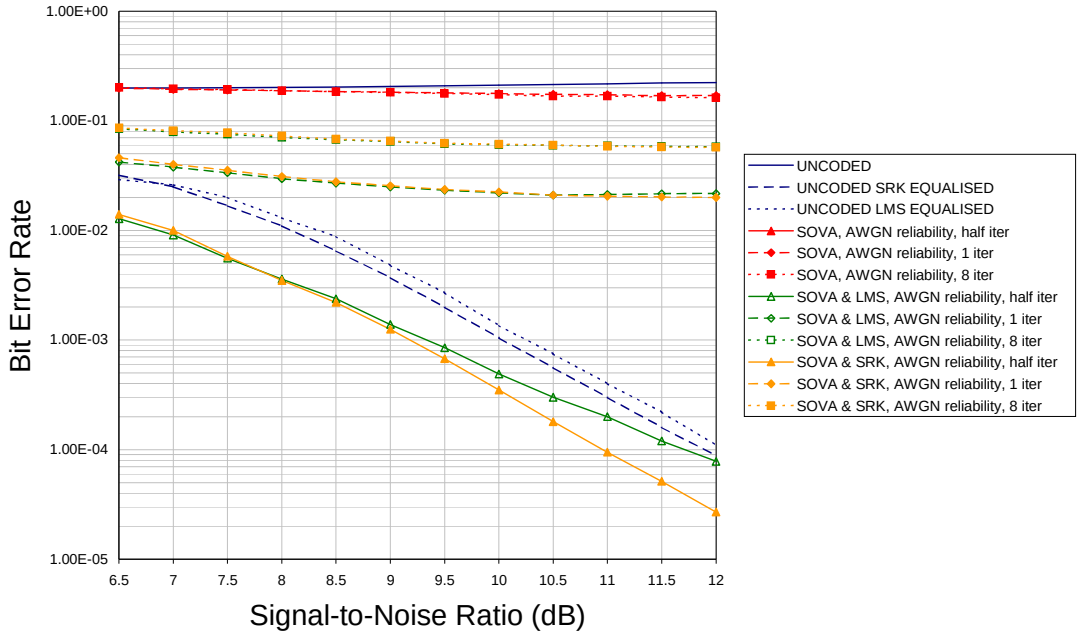


Figure 6.9: Bit Error Rates for the Combination of DFE with $[7;5]_8$ Log-MAP Turbo Codes over 3-tap Static Channel Model, with AWGN Channel Scaling

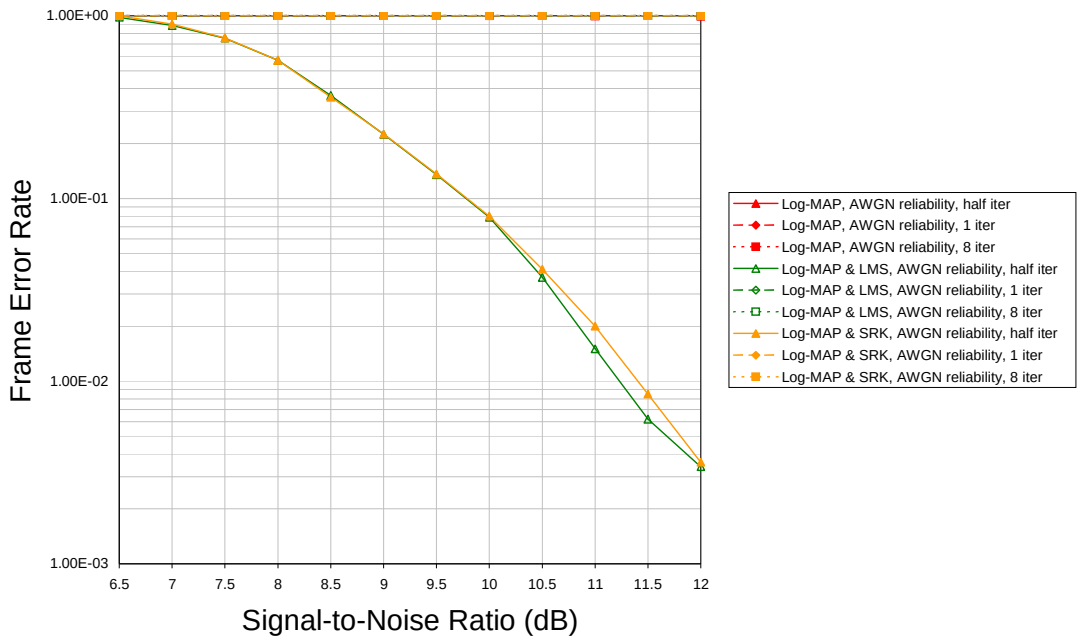


Figure 6.10: Frame Error Rates for the Combination of DFE with $[7;5]_8$ Log-MAP Turbo Codes over 3-tap Static Channel Model, with AWGN Channel Scaling

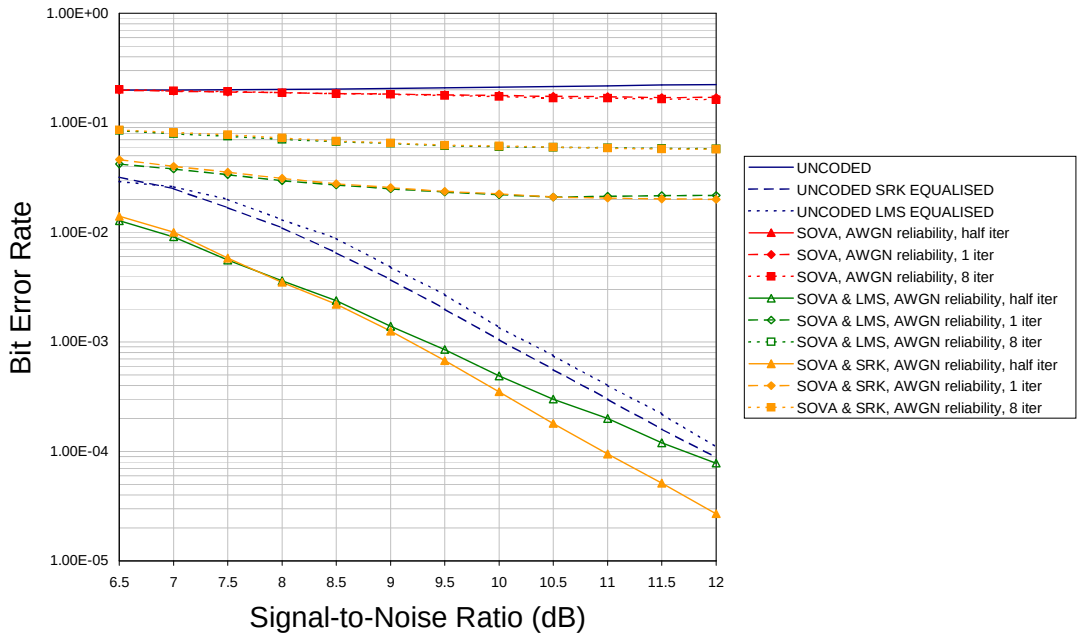


Figure 6.11: Bit Error Rates for the Combination of DFE with $[7;5]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, with AWGN Channel Scaling

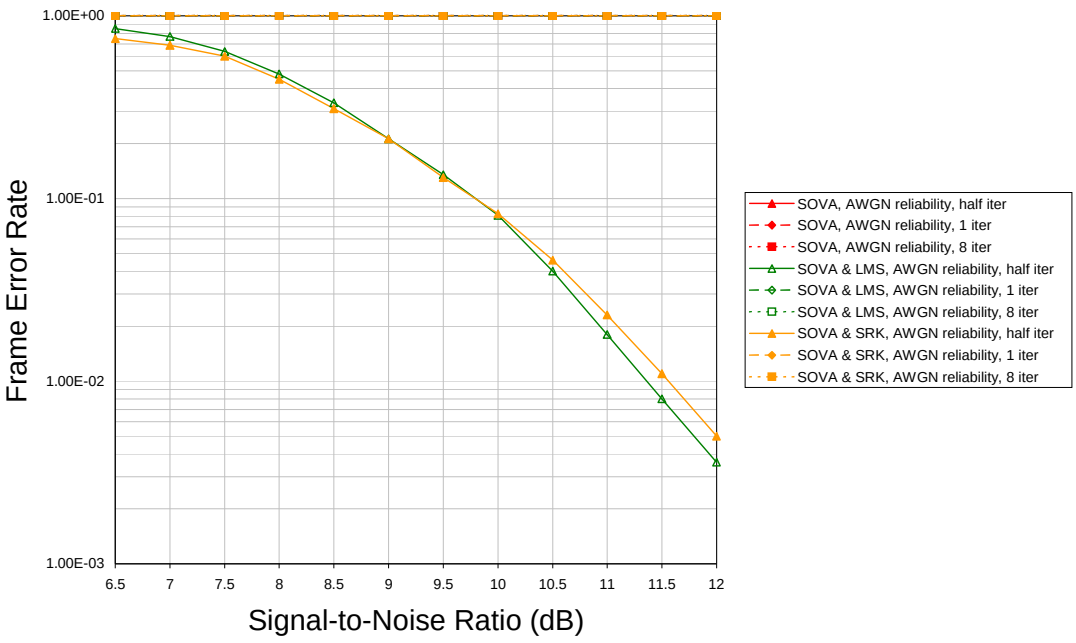


Figure 6.12: Frame Error Rates for the Combination of DFE with $[7;5]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, with AWGN Channel Scaling

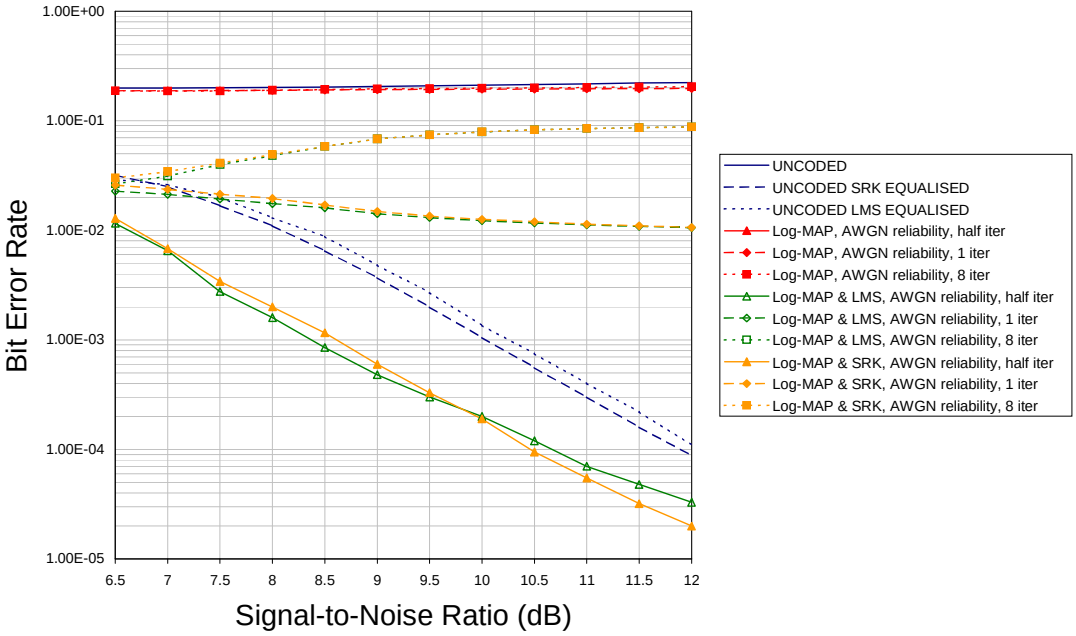


Figure 6.13: Bit Error Rates for the Combination of DFE with $[23;33]_8$ Log-MAP Turbo Codes over Static 3-tap Channel Model, with AWGN Channel Scaling

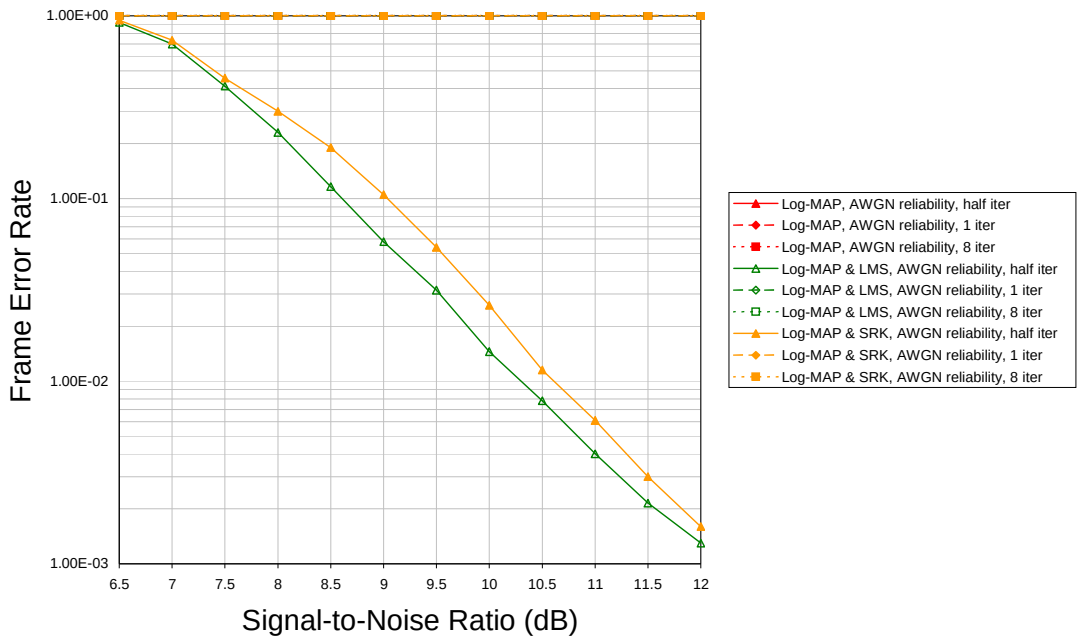


Figure 6.14: Frame Error Rates for the Combination of DFE with $[23;33]_8$ Log-MAP Turbo Codes over Static 3-tap Channel Model, with AWGN Channel Scaling

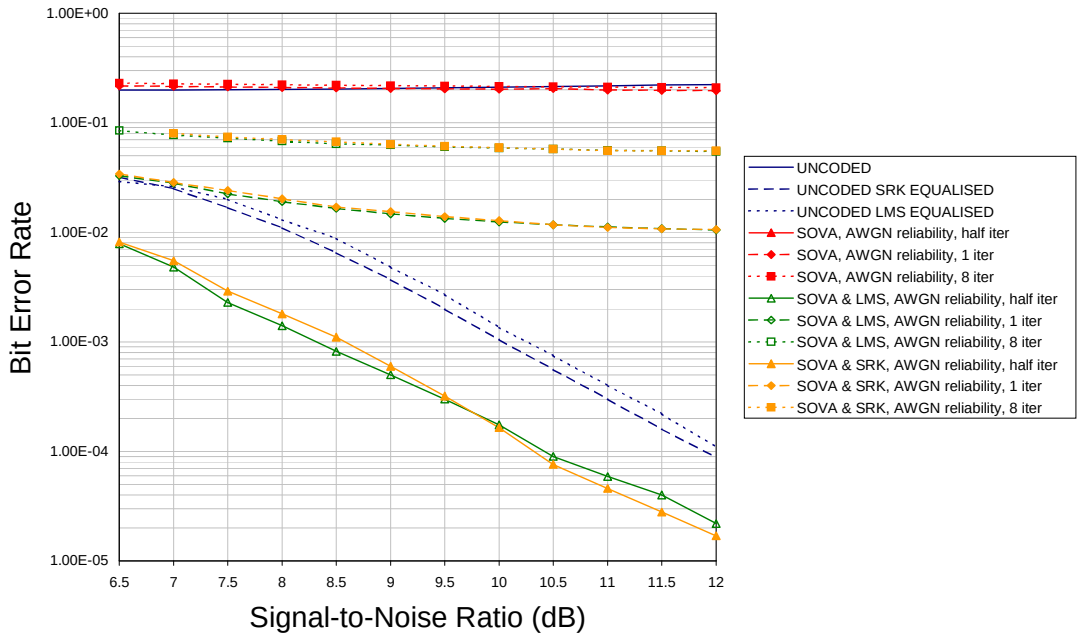


Figure 6.15: Bit Error Rates for the Combination of DFE with $[23;33]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, with AWGN Channel Scaling

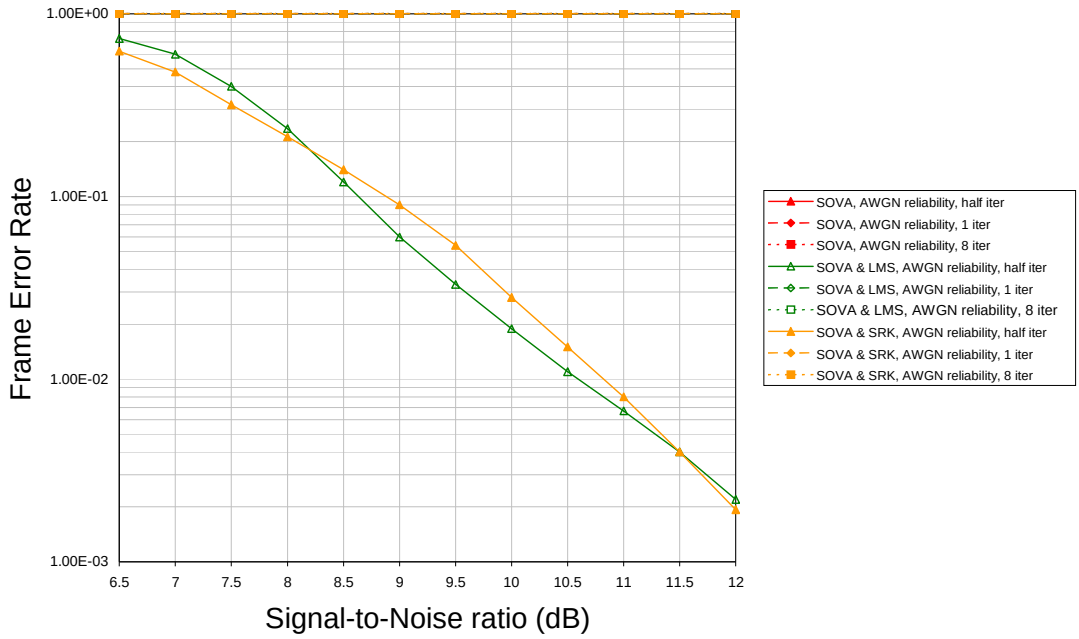


Figure 6.16: Frame Error Rates for the Combination of DFE with $[23;33]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, with AWGN Channel Scaling

6.6.2 Discussion of Time-Invariant Simulations with AWGN Reliability

As can be seen from the results above, the turbo decoder does not perform well when combined with decision feedback equalisers in this manner, with only the Log-MAP turbo decoder outperforming uncoded, equalised, information, and then only slightly and just at very low signal-to-noise ratios. This is because the input to the turbo decoder is virtually binary and may therefore be considered as hard information, causing the reliability decisions created by each component decoder to become increasingly corrupted.

Corroborating this is the fact that the only point at which satisfactory decoded results were produced was after the equivalent of one half-iteration, where reliability information from previous decoders is not included.

Note that the performance difference between the uncoded LMS results and the uncoded SRK results over the static channel is small and that this is carried through to the results of the decoder/equaliser combinations. This appears to be related to the static nature of the channel, where the quicker convergence time of SRK is of limited use. The benefit of this faster convergence can be seen when considering the same combinations over the vehicular channel, where the relative power of the taps is changing much more rapidly.

Where no equalisation was used, the two decoding algorithms produced results that were better than uncoded. However, the lack of proper channel information meant that the improvement was

minor, and did not increase from iteration to iteration as it had under AWGN and Rayleigh fading conditions.

Having said that, there is still a significant amount of information about both the decoding algorithms and the equaliser reactions that can be gained from these results.

To begin with, consider the response of the decoding algorithms to the lack of fading information. A single Log-MAP decoder returns respectable performance, however, the Log-MAP turbo decoder does not react well to the lack of information. It can be seen that, for both component codes with both equalisers, the degradation in performance, after one iteration is very large. After eight iterations, the effect is increased to such an extent that the results are rather bizarre. The performance curves become virtual mirror images of those obtained after one iteration, rising with signal-to-noise ratio until a plateau is reached.

The SOVA algorithm does not undergo such drastic distortion, although clearly it is still heavily affected by the lack of information. After one iteration, the SOVA turbo decoder is severely degraded on comparison with the single SOVA decoder, even more so after eight iterations, but the bit error rate performance still reduces as signal-to-noise ratio increase.

Table 6.1 shows some points of interest obtained from the previous results.

Code	$[7;5]_8$						$[23;33]_8$					
Decoder	Log-MAP			SOVA			Log-MAP			SOVA		
Iteration Number	0.5	1	8	0.5	1	8	0.5	1	8	0.5	1	8
LMS,10^{-2}	6.9dB	-	-	6.86dB	-	-	6.625dB	12dB	-	6.3dB	12dB	-
SRK,10^{-2}	7.04dB	-	-	7dB	-	-	6.69dB	12dB	-	6.3dB	12dB	-
LMS,10^{-4}	12dB	-	-	11.72dB	-	-	10.67dB	-	-	10.44dB	-	-
SRK,10^{-4}	10.9dB	-	-	10.95dB	-	-	10.45dB	-	-	10.32dB	-	-

Table 6.1: Points of Interests for BER at Various Numbers of Iterations for Turbo Codes Combined with DFE Assuming AWGN Channel Information

The table above immediately outlines the fact that iterative decoding of this nature does not work with decision feedback equalisation. The lack of fading information and the fact that the received information has been pre-processed lead to increasingly obvious turbo decoder error. Due to this, it is not possible to review the results of the turbo effect of these decoders in the same way that was done previously, instead the table above is used to review the qualities of the two algorithms after a single pass through a decoder. The results of a half-iteration are therefore considered.

Examining table 6.1, conclusions can be drawn about the effectiveness of the equaliser algorithms and the decoder algorithms. With regard to the equaliser algorithms first, it can be seen that the optimised LMS equaliser outperforms the optimised SRK equaliser at high, voice quality bit error rates, with an equaliser gain of 0.14dB and 0.065dB for the $[7;5]_8$ and $[23;33]_8$ codes respectively, when using Log-MAP decoding. The equivalent gains, using the SOVA decoder are 0.14dB and 0dB.

At lower, data quality bit error rates, the SRK equaliser produces better results, with a gain of 1.1dB for the 4-state code and 0.22dB for the 16-state code for Log-MAP decoding and 0.77dB and 0.12dB for the same codes using SOVA.

Considering the decoding algorithms, SOVA tends to outperform the Log-MAP decoder at both voice quality and data quality bit error rates. At voice quality bit error rates, SOVA shows gains of 0.04dB for both equalisers using the 4-state code, and 0.325dB and 0.39dB for LMS and SRK respectively, when using the 16-state code. This would imply that the Log-MAP algorithm is more sensitive to the lack of decent reliability information received from previous decoders, a fact borne out by the dismal results produced by the decoder when multiple iterations are considered.

Looking at the two component codes, it becomes clear that the $[23;33]_8$ code, with the larger effective free distance is consistently better than the $[7;5]_8$. However, it would be incorrect to state that the performance increase is due to this, as the results considered are from a half-iteration. A more accurate statement would be that the code with the larger minimum Hamming distance was the better performer, as the second parity information was not used during this decode. The 16-state code exhibits gains for both algorithms and both equalisers. For the Log-MAP decoder with the LMS algorithm, the gain at 10^{-2} is 0.275dB. This increases to 1.33dB at 10^{-4}. The same decoder with the SRK equaliser shows gains of 0.35dB and 0.45dB at the same points of interest. For the SOVA decoder, the LMS equaliser and $[23;33]_8$ component code show gains of 0.56dB and 1.28dB at 10^{-2} and 10^{-4} respectively. The same decoder with the SRK equaliser shows gains of 0.7dB and 0.63dB at the same points of interest. These results indicate that the coding gains increase with SNR when LMS is the chosen equaliser algorithm and tend to remain relatively constant with SRK.

To examine the effect of the turbo decoder process, the performance of the decoding algorithms must be compared at signal-to-noise ratios rather than bit error rates. Table 6.2 shows the performance results of the four separate systems at various noise levels.

		$[7;5]_8$ Component Code			$[23;33]_8$ Component Code		
	System	7dB	9dB	12dB	7dB	9dB	12dB
1 Iteration	Log-MAP with LMS	$2.86e^{-2}$	$2.16e^{-2}$	$1.86e^{-2}$	$2.13e^{-2}$	$1.42e^{-2}$	$1.06e^{-2}$
	Log-MAP with SRK	$3.07e^{-2}$	$2.28e^{-2}$	$1.85e^{-2}$	$2.3e^{-2}$	$1.48e^{-2}$	$1.06e^{-2}$
	SOVA with LMS	$3.78e^{-2}$	$2.49e^{-2}$	$2.17e^{-2}$	$2.8e^{-2}$	$1.47e^{-2}$	$1.05e^{-2}$
	SOVA with SRK	$4e^{-2}$	$2.55e^{-2}$	$2e^{-2}$	$2.85e^{-2}$	$1.54e^{-2}$	$1.06e^{-2}$
8 iterations	Log-MAP with LMS	$3.89e^{-2}$	$5.75e^{-2}$	$7.06e^{-2}$	$3.13e^{-2}$	$6.81e^{-2}$	$8.78e^{-2}$
	Log-MAP with SRK	$4.1e^{-2}$	$5.86e^{-2}$	$7.04e^{-2}$	$3.4e^{-2}$	$6.81e^{-2}$	$8.78e^{-2}$
	SOVA with LMS	$7.91e^{-2}$	$6.46e^{-2}$	$5.7e^{-2}$	$7.7e^{-2}$	$6.25e^{-2}$	$5.45e^{-2}$
	SOVA with SRK	$8.1e^{-2}$	$6.55e^{-2}$	$5.71e^{-2}$	$8.9e^{-2}$	$6.36e^{-2}$	$5.53e^{-2}$

Table 6.2: Bit Error Rate Performance of Turbo Codes in Combination with Equalisers, using AWGN Channel Reliability

The results in the table above show some information that is unclear in the figures. For both component codes, the Log-MAP turbo decoder produces better results than the SOVA after one iteration, although the two decoder algorithms appear to converge at higher SNRs, more obviously for the $[23;33]_8$ component codes, regardless of which equaliser update algorithm has been used.

The fact that the Log-MAP decoder reacts so badly to the propagation of errors in *a priori* information means that the SOVA decoder begins to exhibit an improvement over its counterpart once more iterations have been performed. This is evident at the 12dB point only for the 4-state code, whereas, for the 16-state code, it comes into effect at the 9dB point.

Comparing the equaliser update algorithms, it can be seen that the two algorithms performances are quite similar. The LMS algorithm tends to provide slightly better results at lower signal-to-noise ratios, but it appears that the SRK algorithm follows a slightly steeper curve, beginning to show improvements upon the simpler update algorithm at higher SNRs.

Of the two component codes, the 16-state code produces better performance as would be expected due to its larger effective free distance. It is notable, however, that the combination of this code with Log-MAP turbo decoding causes the $[7;5]_8$ code to perform better at lower BERs.

Table 6.3 shows some points of interest for the frame error rate results.

Code	$[7;5]_8$		$[23;33]_8$	
Decoder	Log-MAP	SOVA	Log-MAP	SOVA
LMS,10^{-1}	9.75dB	9.8dB	8.6dB	8.625dB
SRK,10^{-1}	9.75dB	9.8dB	9dB	8.875dB
LMS,5×10^{-3}	11.67dB	11.8dB	10.83dB	11.28dB
SRK,5×10^{-3}	11.8dB	12dB	11.125dB	11.35dB

Table 6.3: Points of Interests for FER at Half Iteration for Turbo Codes Combined with DFE Assuming AWGN Channel Information

As can be seen from the figures showing frame error rates above, the effect of increasing the number of iterations does little for the frame error performance of the receiver systems. In fact, the number of frames with errors is increased as more iterations are performed, with the result that the FER is virtually 1 at all simulated points.

Unlike the bit error rate results, at the lower point of interest, it can be seen that the LMS equaliser results tend to outperform the SRK results, although the two algorithms are very similar.

For the 4-state code results, both equalisers perform equally well at the higher point of interest, reaching 10^{-1} at 9.75dB with the Log-MAP decoder and 9.8dB with the SOVA decoder. At the lower point of interest, the LMS equaliser produces better results with a gain of 0.13dB using Log-MAP and 0.2dB using SOVA. The results for the 16-state code show that the increase in minimum Hamming distance causes the difference between the two equalisers performances to increase at the higher point of interest, with the LMS algorithm showing a gain of 0.4dB using Log-MAP and 0.25dB using SOVA. At 5×10^{-3}, the gain with the Log-MAP decoding algorithm is 0.295dB, whereas the gain with the SOVA decoder is 0.07dB.

As with the bit error performance, and as expected, the $[23;33]_8$ code outperforms the $[7;5]_8$ with coding gains for both decoder algorithms, using both equalisers. Using LMS, the gain at 10^{-1} is 1.15dB for Log-MAP and 1.175dB for SOVA. These reduce to 0.84dB for the Log-MAP and 0.52dB for the SOVA at the lower point of interest. With SRK, the gains are smaller in general. At the higher point of interest, the 16-state code exhibits gains of 0.75dB for the Log-MAP and 0.925dB for the SOVA, further reducing to 0.675dB for Log-MAP and 0.65dB for SOVA at the higher point of interest.

The decoder gains are small too, especially for the $[7;5]_8$ code. With either equaliser, the gain obtained at the higher point of interest by Log-MAP with this code is just 0.05dB. This increases slightly at the lower point of interest with gains of 0.13dB using LMS and 0.2dB using SRK. The $[23;33]_8$ code is no better, with gains at the higher point of interest of 0.025dB using LMS and 0.125dB using SRK. Again, these increase at 5×10^{-3} with a gain for Log-MAP with LMS of 0.45dB and 0.225dB with SRK.

Under these conditions, it is obvious that the iterative, turbo effect is of no benefit. Both decoders suffer from the lack of reliability information and the benefits that were apparent in previous results are therefore not present. In these circumstances, the SOVA decoder produces better results than

the Log-MAP algorithm where bit error rate is of primary concern and produces frame error performance very close to that of Log-MAP.

To obtain better turbo results, it was suggested that channel reliability should be omitted as the fact that the data supplied to the turbo decoder is no longer in its raw, received, state implies that there is little use, or need for this scaling factor. Incorrect channel information may also cause the component decoders to produce unreliable extrinsic information, further reducing turbo decoder performance. The simulations carried out above were therefore repeated without this factor.

6.6.3 Time-Invariant Simulations with Channel Reliability of 1

Identical simulations to those above were then run. These simulations used a channel reliability set to 1, rather than the AWGN reliability that was used previously.

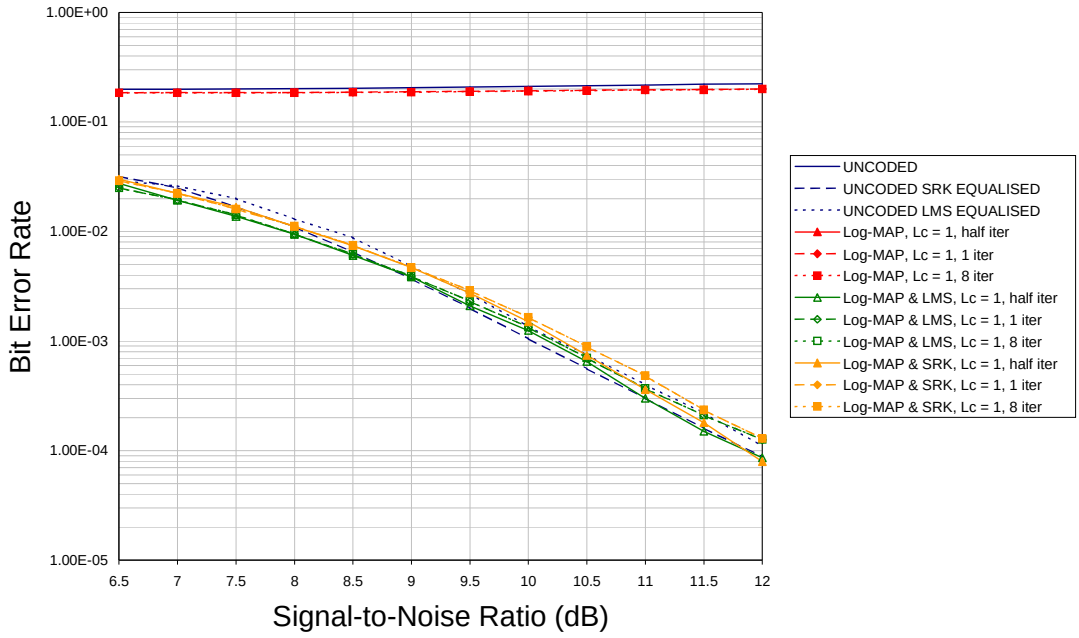


Figure 6.17: Bit Error Rates for the Combination of DFE with $[7;5]_8$ Log-MAP Turbo Codes over Static 3-tap Channel Model, with $L_c = 1$

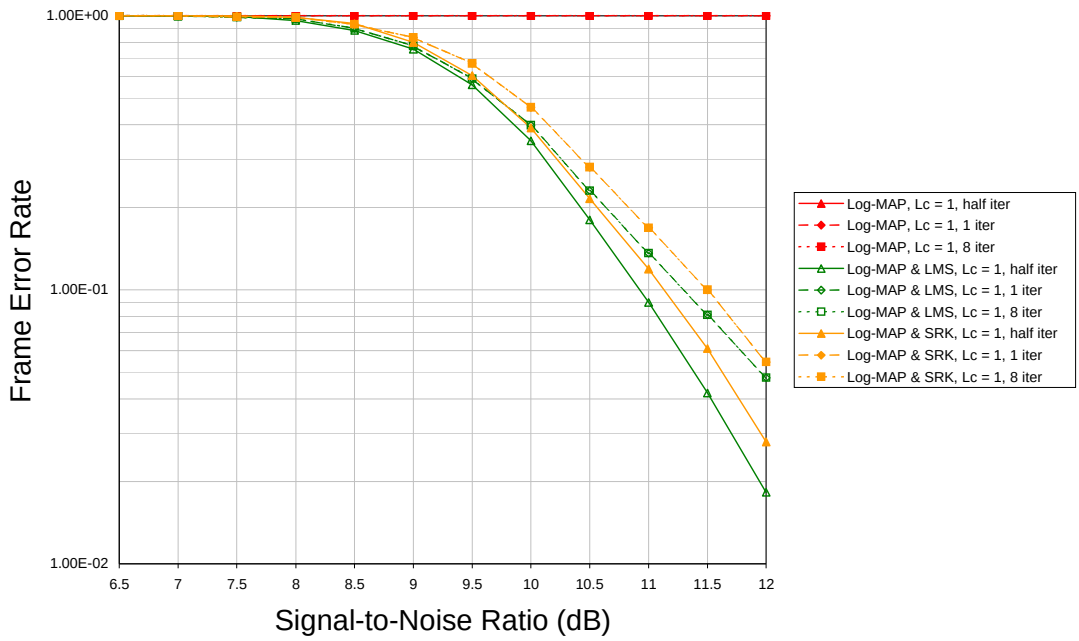


Figure 6.18: Frame Error Rates for the Combination of DFE with $[7;5]_8$ Log-MAP Turbo Codes over Static 3-tap Channel Model, $L_c = 1$

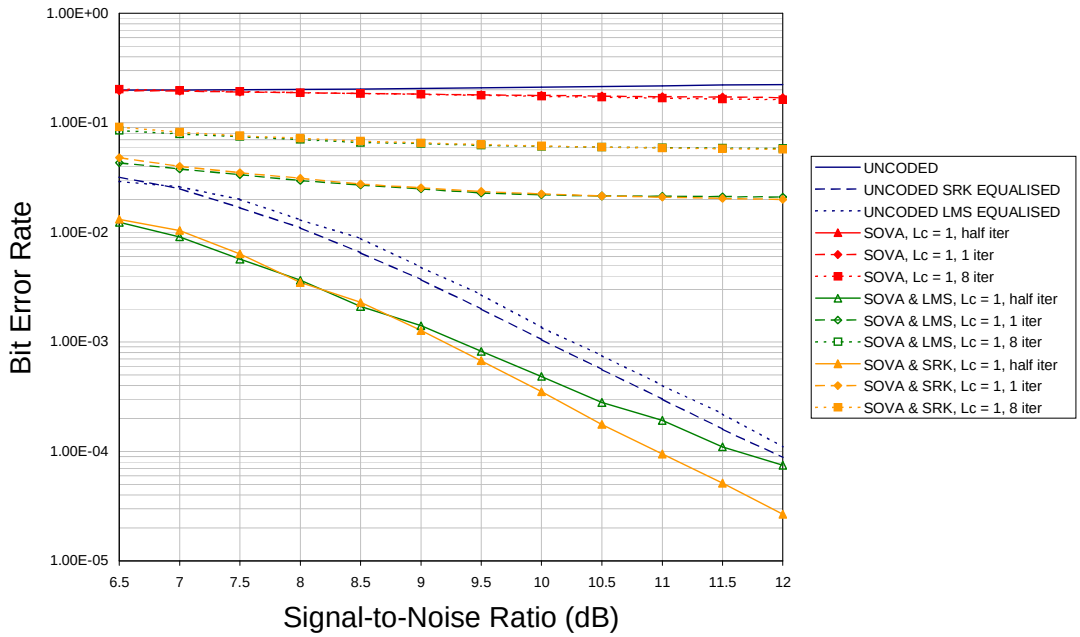


Figure 6.19: Bit Error Rates for the Combination of DFE with $[7;5]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, $L_c = 1$

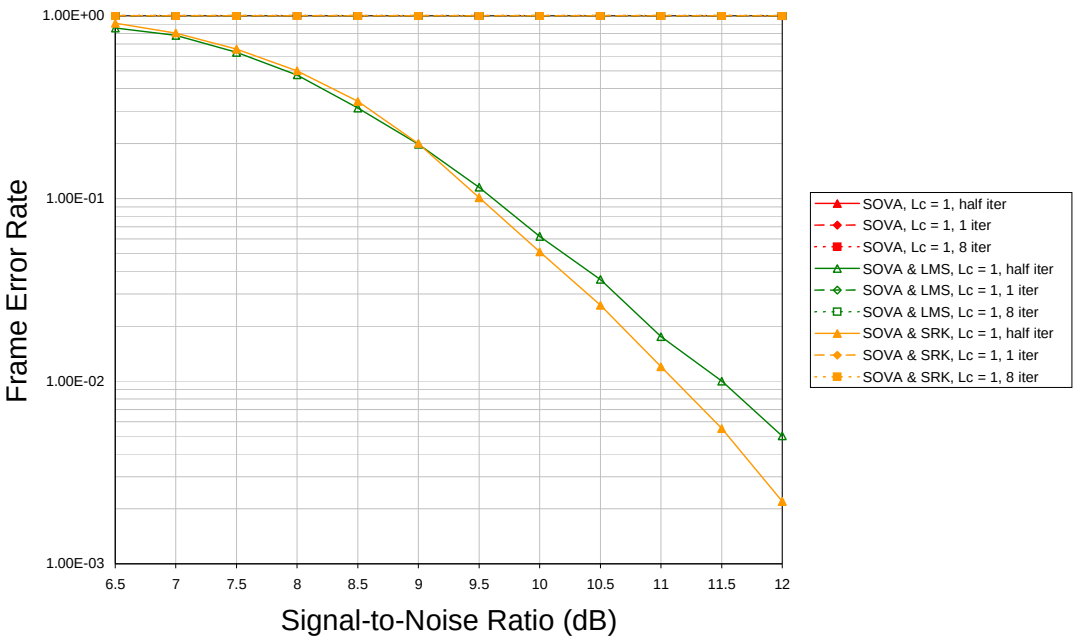


Figure 6.20: Frame Error Rates for the Combination of DFE with $[7;5]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, $L_c = 1$

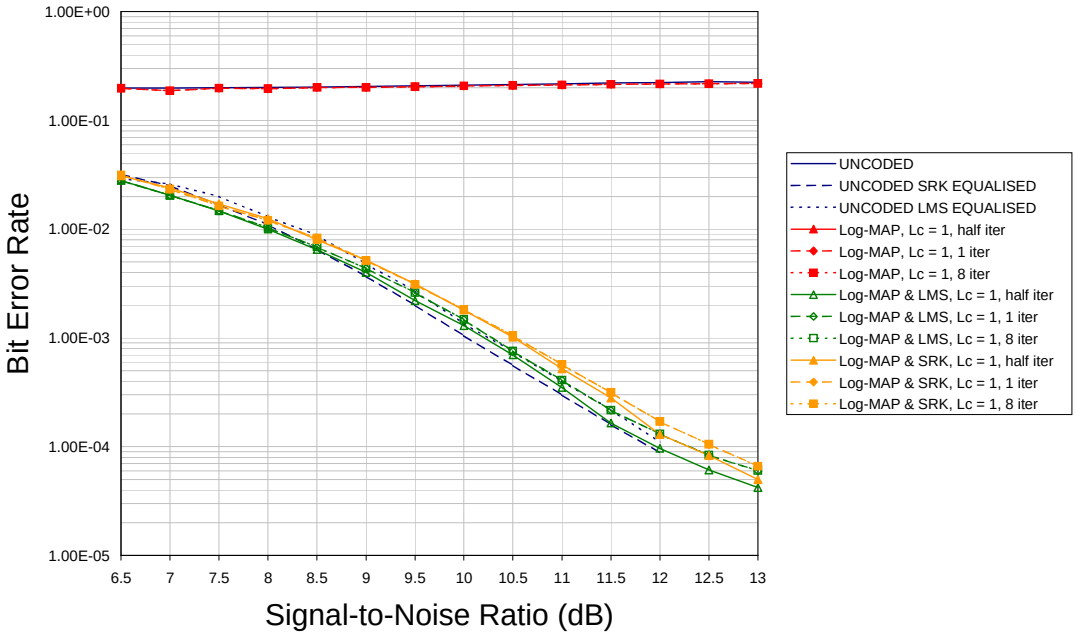


Figure 6.21: Bit Error Rates for the Combination of DFE with $[23;33]_8$ Log-MAP Turbo Codes over Static 3-tap Channel Model, $L_c = 1$

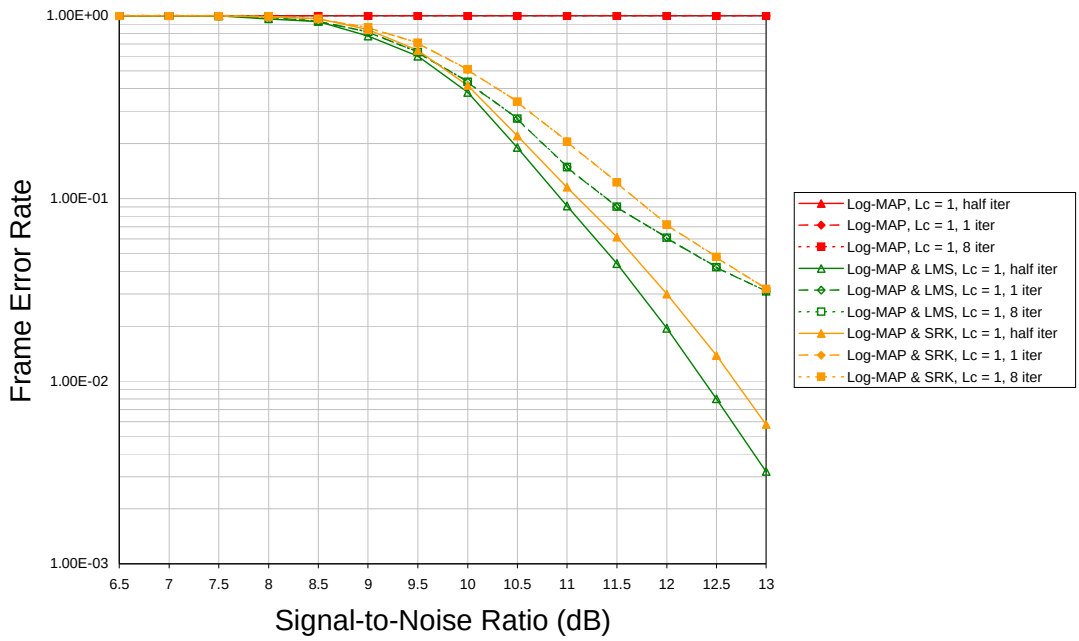


Figure 6.22: Frame Error Rates for the Combination of DFE with $[23;33]_8$ Log-MAP Turbo Codes over Static 3-tap Channel Model, with $L_c = 1$

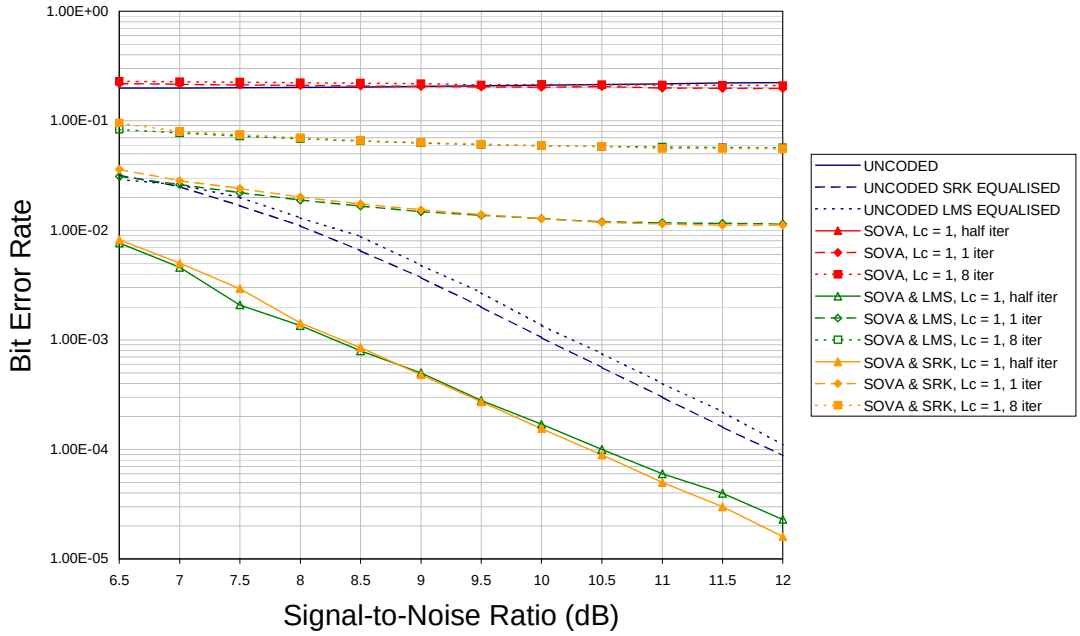


Figure 6.23: Bit Error Rates for the Combination of DFE with $[23;33]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, $L_c = 1$

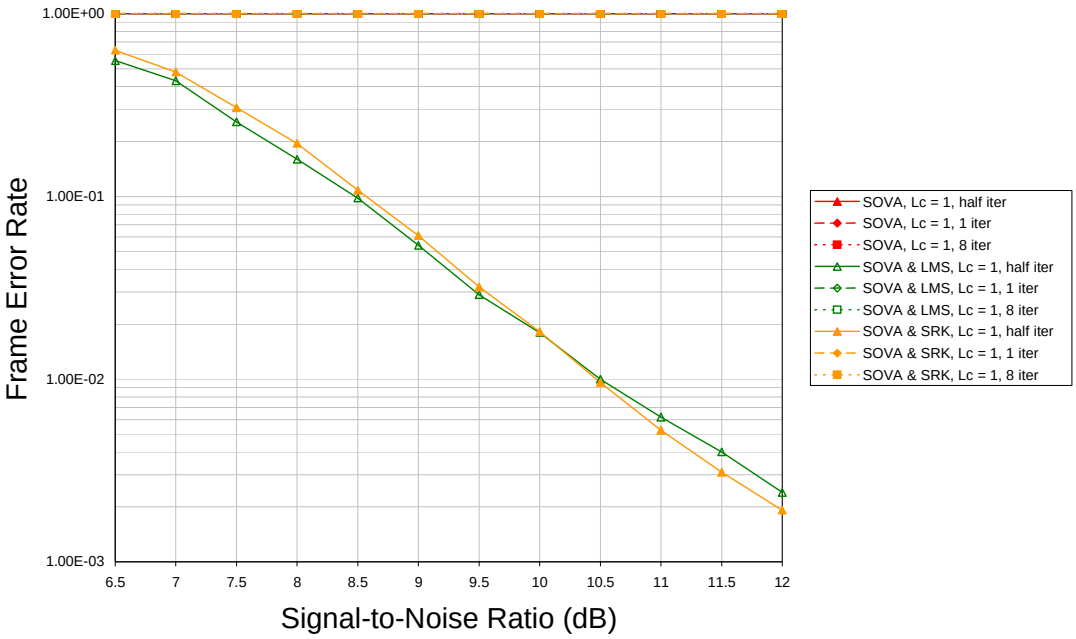


Figure 6.24: Frame Error Rates for the Combination of DFE with $[23;33]_8$ SOVA Turbo Codes over Static 3-tap Channel Model, with $L_c = 1$ (Note: Results that are not visible follow SOVA SRK 8 iteration results)

6.6.4 Discussion of Time-Invariant Simulations with Channel Reliability of 1

Where the first set of results showed the effect of assuming AWGN channel information at the receiver, these results show how the decoders react when it is assumed that the received information is pre-processed by the equaliser and therefore no scaling factor is used.

On first impressions, the effect on the SOVA decoder is unnoticed. However, the Log-MAP iterative performance has improved greatly from the previous results, although any performance improvements expected from iterative decoding are still non-existent.

As before, table 6.4 below shows the bit error rates at different signal-to-noise ratios along the decoded curves

		$[7;5]_8$ Component Code			$[23;33]_8$ Component Code		
	System	7dB	9dB	12dB	7dB	9dB	12dB
1 Iteration	Log-MAP with LMS	$1.94e^{-2}$	$3.91e^{-3}$	$1.26e^{-4}$	$2.05e^{-2}$	$4.37e^{-3}$	$1.31e^{-4}$
	Log-MAP with SRK	$2.23e^{-2}$	$4.72e^{-3}$	$1.29e^{-4}$	$2.33e^{-2}$	$5.16e^{-3}$	$1.7e^{-4}$
	SOVA with LMS	$3.78e^{-2}$	$2.48e^{-2}$	$2.1e^{-2}$	$2.61e^{-2}$	$1.47e^{-2}$	$1.14e^{-2}$
	SOVA with SRK	$3.99e^{-2}$	$2.54e^{-2}$	$2e^{-2}$	$2.83e^{-2}$	$1.54e^{-2}$	$1.11e^{-2}$
8 iterations	Log-MAP with LMS	$1.94e^{-2}$	$3.91e^{-3}$	$1.26e^{-4}$	$2.05e^{-2}$	$4.37e^{-3}$	$1.31e^{-4}$
	Log-MAP with SRK	$2.23e^{-2}$	$4.72e^{-3}$	$1.29e^{-4}$	$2.33e^{-2}$	$5.16e^{-3}$	$1.7e^{-4}$
	SOVA with LMS	$7.87e^{-2}$	$6.44e^{-2}$	$5.8e^{-2}$	$7.73e^{-2}$	$6.26e^{-2}$	$5.6e^{-2}$
	SOVA with SRK	$8.17e^{-2}$	$6.54e^{-2}$	$5.7e^{-2}$	$7.98e^{-2}$	$6.33e^{-2}$	$5.52e^{-2}$

Table 6.4: Bit Error Rate Performance of Turbo Codes in Combination with Equalisers, using Channel Reliability of 1

The main difference between this table and table 6.2 is that the results for the Log-MAP decoding algorithm reduce with signal-to-noise ratio. Comparing the two sets of simulations shows that where a channel reliability of one has been used, the turbo decoder with this algorithm produces better results. The results remain virtually unchanged for the SOVA decoder from the previous set of simulations, but the Log-MAP results show a definite downward trend.

It is interesting to note that the results from one iteration to eight iterations remain virtually unchanged, whereas the SOVA turbo decoder shows a marked reduction in performance in much the same way as it did where AWGN channel reliability was used.

With regards to the equaliser update algorithms, the Log-MAP decoder performs better with LMS at all points considered. The results show that although LMS is a better algorithm for the lower SNR range, it is consistently beaten by SRK at higher SNRs when in combination with SOVA. The performance increase is only small however.

Considering the two component codes, with SOVA turbo decoding, the $[23;33]_8$ component code is more successful, however, the $[7;5]_8$ code performs better when Log-MAP is used by the decoder.

Table 6.5 shows the points of interest for bit error rate performance.

Code	$[7;5]_8$						$[23;33]_8$					
Decoder	Log-MAP			SOVA			Log-MAP			SOVA		
Iteration Number	0.5	1	8	0.5	1	8	0.5	1	8	0.5	1	8
LMS 10^{-2}	7.95dB	7.95dB	7.95dB	6.86dB	-	-	8dB	8.05dB	8.05dB	6.25dB	12dB	-
SRK 10^{-2}	8.125dB	8.125dB	8.125dB	7.05dB	-	-	8.25dB	8.25dB	8.25dB	6.25dB	12dB	-
LMS 10^{-4}	11.875dB	12.25dB	12.25dB	11.64dB	-	-	11.95dB	12.29dB	12.29dB	10.5dB	-	-
SRK 10^{-4}	11.875dB	12.2dB	12.2dB	10.95dB	-	-	12.29dB	12.56dB	12.56	10.4dB	-	-

Table 6.5: Points of Interests for BER at Various Numbers of Iterations for Turbo Codes Combined with DFE Assuming No Channel Information Scaling Factor

On closer inspection, the omission of the channel scaling factor has in fact reduced the performance of the Log-MAP decoding algorithm. The reliability values passed between decoders have become reduced to such an extent that they play almost no role in the decoding process, resulting in what appears to be an improvement upon the previous results.

It is obvious that the Log-MAP turbo decoder has produced gains in comparison to those obtained with AWGN results as the results at one iteration and 8 iterations are improvements upon the previous results. However, when examining the effect of the omission of the channel reliability scaling factor on the half-iteration results it is apparent that this is detrimental to the basic decoder algorithm.

Comparing the Log-MAP, half-iteration, results of table 6.3 with those of table 6.1, shows that the omission of channel information has caused losses. The $[7;5]_8$ decoder requires 1.05dB using LMS and 1.085dB with SRK more signal-to-noise ratio to reach voice quality bit error rates. The $[23;33]_8$ decoder requires 1.375dB more and 1.56dB more for using the same equalisers respectively. At the lower point of interest, the $[7;5]_8$ code in combination with the LMS equaliser produces a 0.125dB improvement upon the results obtained using AWGN channel reliability, but it is the only combination that does. With SRK, the same decoder requires a signal-to-noise ratio 0.975dB greater with no channel reliability, while the $[23;33]_8$ Log-MAP decoder requires SNRs 1.28dB and 1.84dB greater for LMS and SRK respectively.

The extrinsic information produced when using more than one decoder has a much less detrimental effect than previously, although performance is still reduced from that of a half-iteration. At the higher point of interest, the effect is almost nil, only the 16-state code in combination with LMS suffers at all, with performance reduced by 0.05dB. The effect is clearer at the lower point of interest but does not get worse as the number of iterations increases. The $[7;5]_8$ Log-MAP decoder with LMS requires a SNR 0.375dB greater than for a half-iteration whereas the same decoder combined with SRK requires 0.325dB more. The same combinations, with the $[23;33]_8$ code produce similar losses, 0.34dB with LMS and 0.27dB with SRK.

With the Log-MAP algorithm, the effect of this lack of scaling factor appears to be greater on the 16-state code. Where it was the better performer with AWGN channel reliability, it is now outperformed by the 4-state code. Of the two equaliser algorithms, the LMS produces better results at all points of interest, however it is likely that this is more due to the decoder problems than the equaliser.

The SOVA decoder produces results very similar to those obtained using AWGN channel reliability. That is to say, that the omission of channel information and scaling did not improve the turbo decoder results, but on the other hand, the results of the half-iteration remained virtually unchanged as well. As the SOVA decoder without extrinsic information functions as a Viterbi decoder when the channel amplitude is assumed to be constant, this is to be expected. The results vary from those previously by no more than 0.08dB at any point. As before with this decoder, the 16-state code, with higher minimum Hamming distance is the better performer and as before, combination with LMS produces better results for both component codes at voice quality bit error rates while combination with SRK produces better results at data quality bit error rates.

As the absence of a scaling factor has had little to no effect upon the SOVA results, this algorithm again produces better performance over Log-MAP when the decoder only performs a half-iteration, with gains for the $[7;5]_8$ code of 1.09dB, using LMS, and 1.12dB, with SRK, at voice quality BERs and 0.235dB in combination with LMS and 0.925dB with SRK. The 16-state $[23;33]_8$ SOVA decoder exhibits gains of 1.75dB with LMS and 2dB with SRK at 10^{-2} and 1.45dB with LMS and 1.89dB with SRK at a BER of 10^{-4}. The gain produced with the 16-state code is larger as is the gain produced with the SRK equaliser. Perhaps another piece of evidence to suggest that the Log-MAP algorithm is more adversely affected by incorrect channel information as was suggested earlier in the thesis.

Table 6.6 shows some points of interest for the frame error performance results.

Code	$[7;5]_8$				$[23;33]_8$			
Decoder	Log-MAP		SOVA		Log-MAP		SOVA	
Iteration Number	0.5	Turbo	0.5	turbo	0.5	turbo	0.5	turbo
LMS,10^{-1}	10.92dB	10.3dB	9.5dB	-	10.95dB	11.4dB	8.45dB	-
SRK,10^{-1}	11.125dB	11.5dB	9.6dB	-	11.1dB	11.7dB	8.55dB	-
LMS,5×10^{-2}	11.375dB	11.95dB	10dB	-	11.4dB	12.3dB	9.05dB	-
SRK,5×10^{-2}	11.625dB	12.05dB	10.2dB	-	11.64dB	12.45dB	9.17dB	-
LMS,5×10^{-3}	-	-	11.55dB	-	12.75dB	-	11.25dB	-
SRK,5×10^{-3}	-	-	12dB	-	13.1dB	-	11.05dB	-

Table 6.6: Points of Interests for FER at Various Numbers of Iterations for Turbo Codes Combined with DFE Assuming No Channel Information

As with the bit error rate performance, the frame error rate for Log-MAP turbo decoding is an improvement upon that previously obtained. However, the effect on the algorithm in general, as shown by the half-iteration results is again the opposite. Also as before, the SOVA algorithm remains largely unchanged by the omission of channel information, with slight improvement throughout the range of points of interest.

Looking at the Log-MAP decoder first and comparing it with the results obtained where AWGN channel information and scaling was assumed, it can be seen that the frame error rate performance of the half-iteration results are reduced. The 4-state Log-MAP decoder does not reach a FER of 5×10^{-3} and reaches 10^{-1} at a signal-to-noise ratio 1.17dB more than with AWGN scaling where LMS is the equaliser algorithm of choice and 1.375dB where SRK is used. The $[23;33]_8$ Log-MAP decoder suffers even more, with losses of 2.35dB using LMS and 2.1dB using SRK at 10^{-1}. This decoder does however reach the third point of interest, 5×10^{-3}, albeit with a SNR 1.92dB higher than previous results with LMS and 1.975dB with SRK. The second point of interest is included in table 6.6 to emphasise the effect of this omission in scaling. The point at which this is reached for both Log-MAP decoders, using both equalisers, is similar to that required for decoders using AWGN channel scaling to reach the third point of interest, a whole order of magnitude difference in results.

Up to the second point of interest, the two Log-MAP decoders perform in a very similar way to each other. For a half-iteration, the results are very close, but for those results that use multiple decoder iterations, the 4-state code consistently outperforms the 16-state version.

Considering the SOVA results, the difference between the results using AWGN channel scaling and those using no scaling are small, as with the BER results. They are again a slight improvement however, with gains at a FER of 10^{-1} of 0.3dB with LMS and 0.2dB using SRK for the 4-state code and 0.175dB and 0.325dB using the same equalisers respectively for the $[23;33]_8$ code. At a FER of 5×10^{-3}, the gains made by omitting the channel scaling factor are 0.25dB and 0dB with LMS and SRK respectively for the 4-state code and 0.03dB and 0.3dB with the same equalisers and the 16-state code.

Comparing the results of the two decoding algorithms shows that the Log-MAP decoder outperforms its SOVA counterpart when more than a half-iteration is performed, however, after one half-iteration, SOVA performs much better, with either equalisation technique and either component code. At the lower point of interest, the 4-state code using SOVA has a gain of 1.42dB over Log-MAP using LMS and 1.525dB using SRK. The 16-state code produces gains of 2.5dB and 2.55dB for the same equalisers at the same FER. At the second point of interest, 5×10^{-2}, the $[7;5]_8$ code produces gains of 1.375dB using LMS and 1.425dB using SRK. The $[23;33]_8$ code produces gains, at this FER, of 2.35dB and 2.47dB for LMS and SRK respectively. The 4-state Log-MAP decoder does not reach 5×10^{-3}, however, the 16-state code produces gains of 1.5dB and 2.05dB at this point of interest, showing that the SOVA decoder is the better performer throughout, under these circumstances.

Frame error performance here again shows that the LMS is generally the better performer with SOVA decoding, with gains of 0.1dB for both component codes at 10^{-1}, 0.2dB and 0.12dB at 5×10^{-2}

and 0.45dB for the $[7;5]_8$ code at 5×10^{-3}. SRK produces a gain of 0.2dB at this final point of interest with the 16-state decoder.

For Log-MAP, at all measured points of interest, the LMS algorithm is again the superior performer with gains at 10^{-1} of 0.25dB for the 4-state and 0.15dB for the 16-state code, 0.25dB and 0.24dB for the same codes at 5×10^{-2} and 0.35dB for the 16-state code at 5×10^{-3}.

In general then, looking at both sets of results, it can be seen that the two decoding algorithms suffer greatly when used in turbo codes from the lack of accurate channel information and scaling. Where AWGN channel information is used, after one iteration, the two turbo decoders produce similar results, however, after 8 iterations, the SOVA decoder produces better results, retaining a reduction in bit error rate as signal-to-noise ratio increase, although only just. The Log-MAP algorithm is far more greatly affected, with results becoming worse as signal-to-noise ratio increases. Where only one half-iteration is used and the problems associated with the propagation of errors through extrinsic information are removed, the two decoding algorithms perform well and in a very similar way. This highlights the fact that the Log-MAP algorithm is of little benefit when used as a conventional decoder and that it is heavily reliant on previous decoders abilities to produce results. The SOVA decoder, devoid of any extrinsic information performs as a Viterbi decoder would be expected to.

Where channel scaling was omitted completely, the Log-MAP turbo decoder was not as heavily affected by errors in extrinsic information. The decoder results did not improve with increases in iterations, however, they did at least produce a performance improvement on those using AWGN channel scaling. The results did affect the algorithm though, the half-iteration results were severely reduced on comparison with those obtained using AWGN channel reliability.

As for the equalisers, the better performer in terms of bit error rate depended on the rate concerned. At voice quality BERs, the LMS algorithm was the better performer, but only slightly. At high signal-to-noise ratios, the SRK algorithm generally performed better, although this was in question where no scaling was used for Log-MAP. In terms of frame error rate, the LMS algorithm tended to produce better results at all points of interest.

Of the two component codes, the $[23;33]_8$ was affected more than the $[7;5]_8$, producing better results as would be expected where AWGN information was used to scale the input to the decoder and where only half an iteration was performed. The effect of increasing the number of iterations was also seen more clearly in this code. It returned better results the 4-state code for 1 iteration, but worse for eight iterations, suggesting that the error propagation problems were more important here. It was also more adversely affected by the lack of fading information at the Log-MAP decoder, with the SOVA decoder gains being larger for this code than for the 4-state version.

6.6.5 16-state Turbo Code Indoor Channel Simulations

The next group of simulations examine the effects caused by the indoor office environment on the 16-state turbo code used previously in this chapter. The code and components used remain unchanged. However, this channel and those that follow are time-varying, as described in chapter 3.

Both the LMS and SRK equalisation update methods were used to counteract the ISI encountered during transmission and channel reliability was considered in the same way as before, comparing the use of AWGN channel reliability scaling and no channel reliability scaling on the same combinations. Simulations were performed for turbo codes using SOVA and Log-MAP component decoders. The results of the indoor channel are presented below.

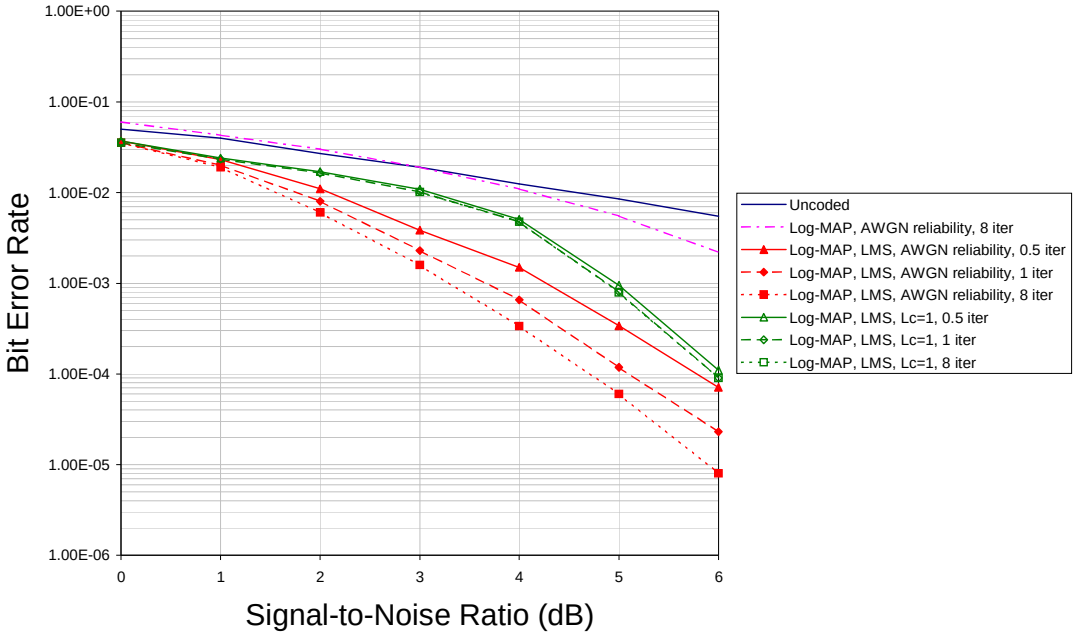


Figure 6.25: BER for Log-MAP Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Indoor Channel

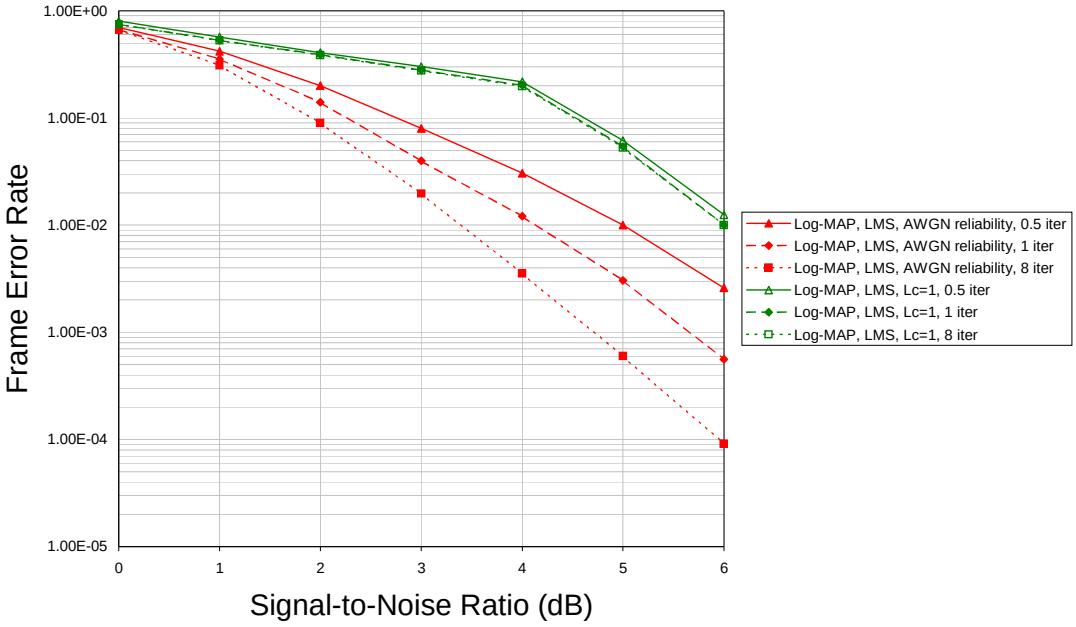


Figure 6.26: FER for Log-MAP Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Indoor Channel

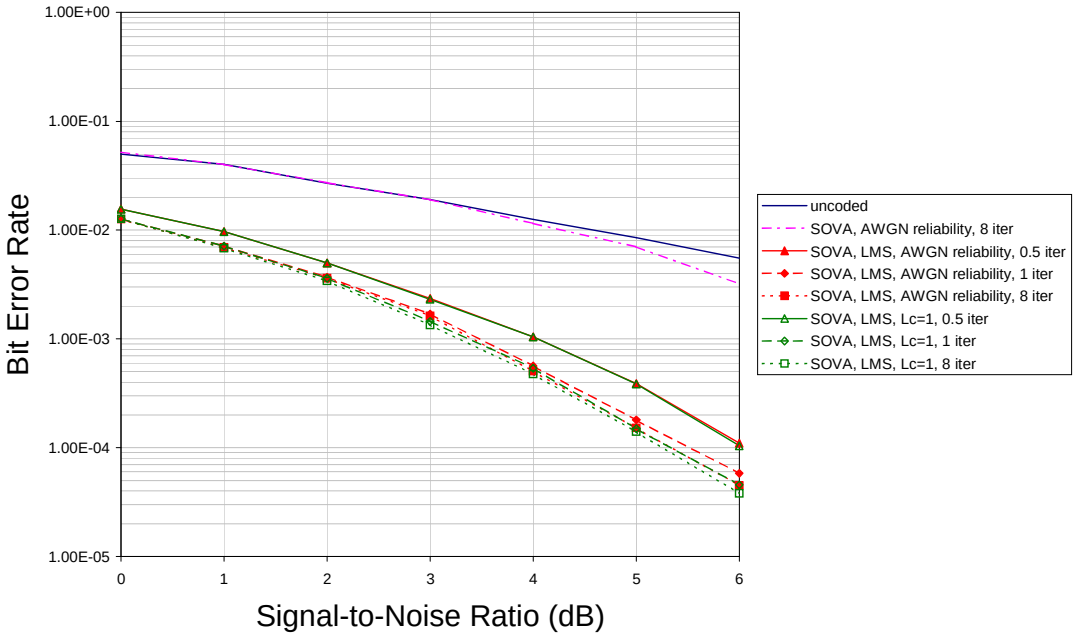


Figure 6.27: BER for SOVA Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Indoor Channel

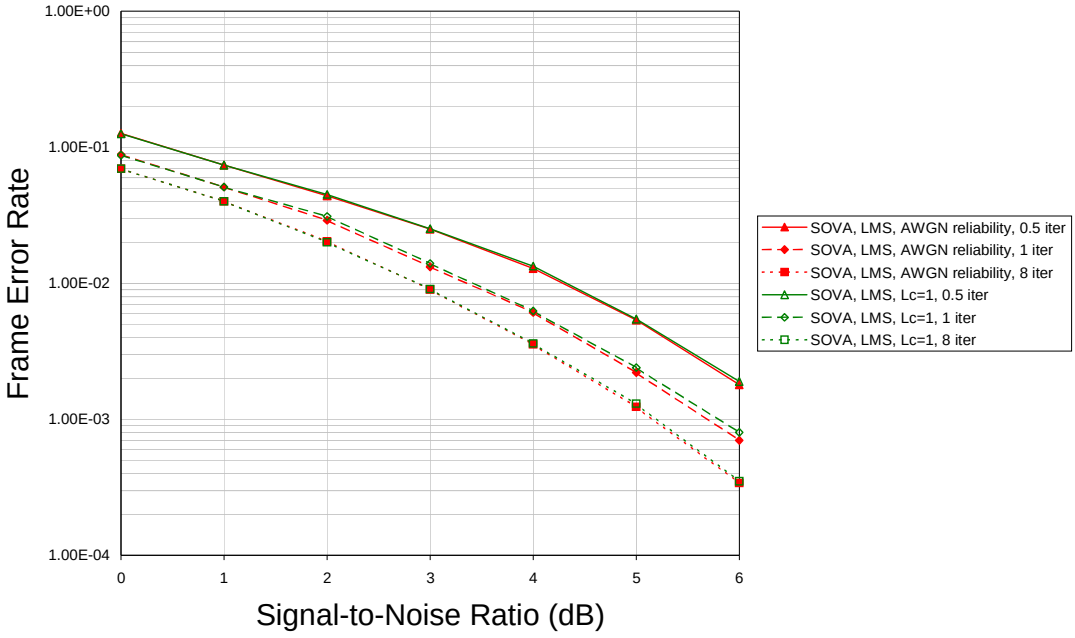


Figure 6.28: FER for SOVA Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Indoor Channel

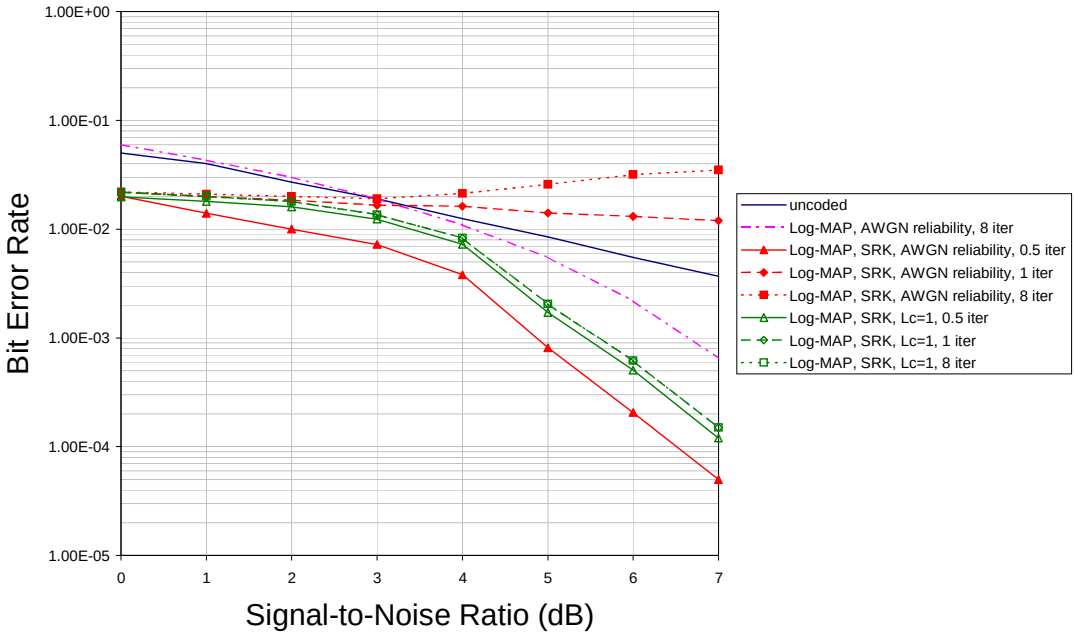


Figure 6.29: BER for Log-MAP Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Indoor Channel

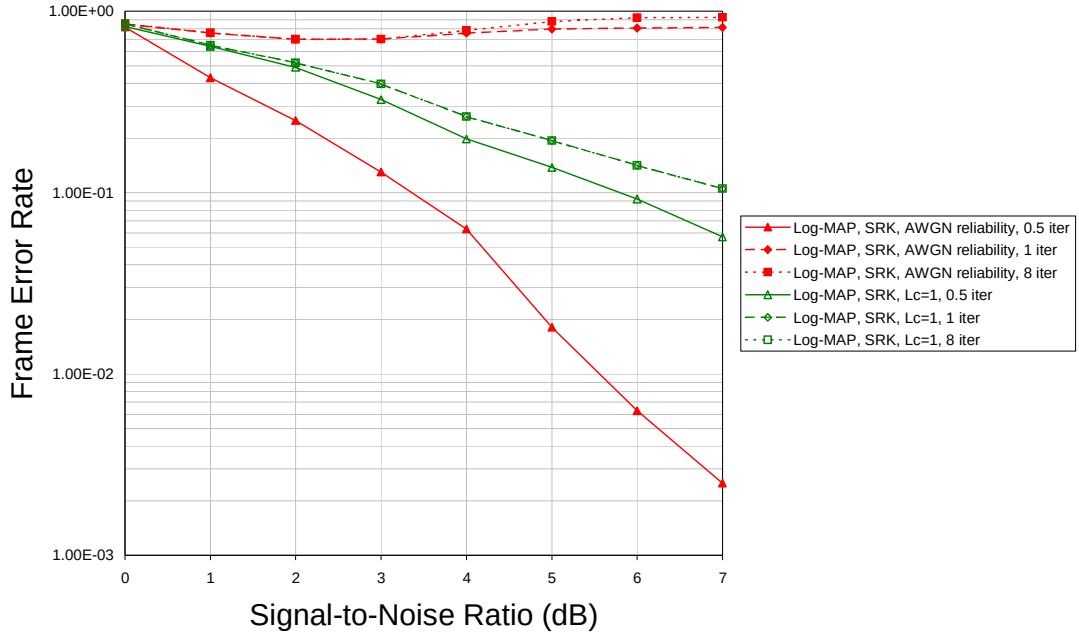


Figure 6.30: FER for Log-MAP Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Indoor Channel

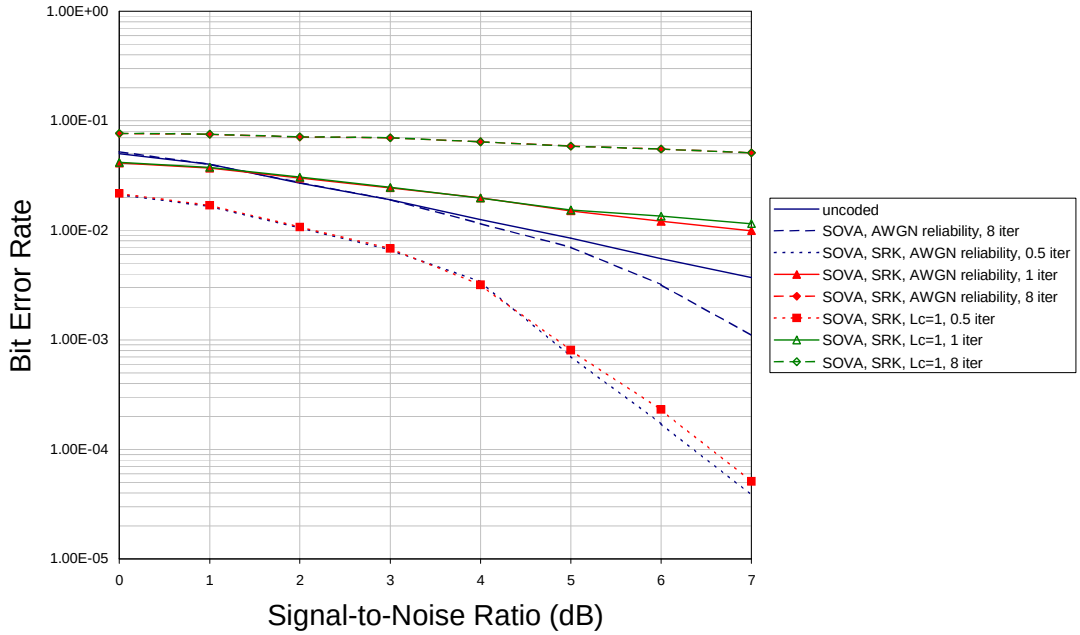


Figure 6.31: BER for SOVA Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Indoor Channel

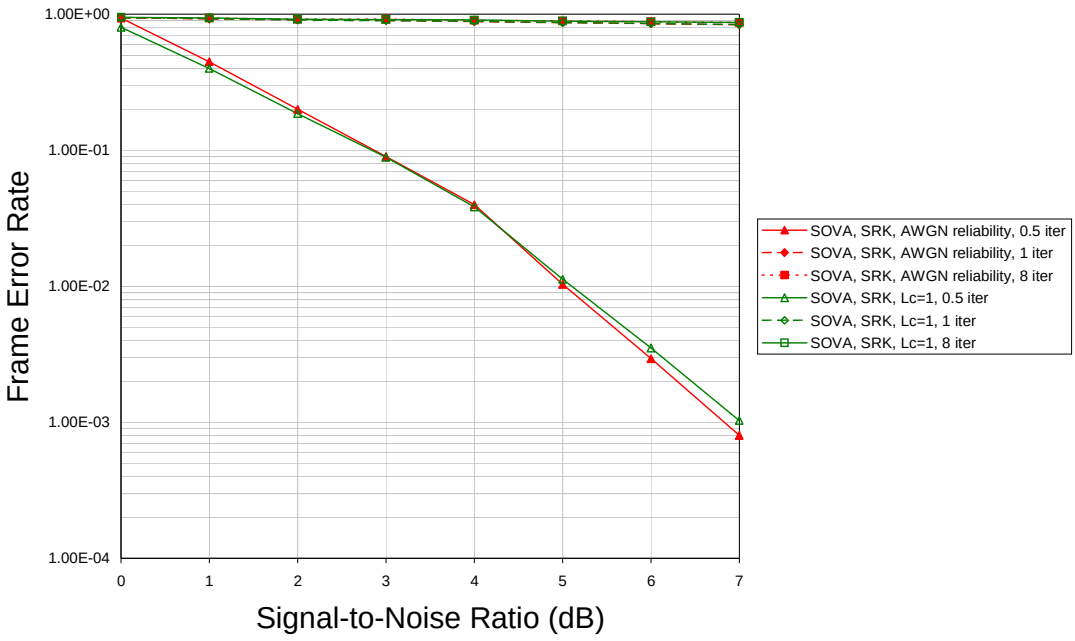


Figure 6.32: FER for SOVA Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Indoor Channel

6.6.6 Discussion of Indoor Channel Simulation Results

Elements of the AWGN results in chapter 5 are present within some of the plots above, as well as elements of the results found with the time-invariant channel earlier in this chapter. To a certain extent, this is to be expected as the ETSI indoor channel uncoded results are similar to AWGN uncoded results. The most interesting aspect of these results is the improvements made in some instances by increasing the number of iterations, which were not obtained in the time-invariant simulations. These effects and others will be discussed here.

Figures 6.25 and 6.27 show effects on the BER of the combination of turbo codes with the LMS equaliser. In both of these figures, it is evident that the turbo effect is in operation. Results improve for both decoding algorithms as the number of iterations increase. In the case of Log-MAP decoding, the increase with AWGN channel scaling produces improvements that are comparable with AWGN simulations. Using no channel scaling, improvements are visible, but are only small and become almost non-existent after one iteration. Overall, the decoder/equaliser combination produces better results with AWGN scaling than when no scaling factor is used, although the results with Lc = 1 begin to curve steeply downwards at higher SNRs and it is possible that these results will produce an improvement on those obtained with AWGN channel scaling later. The SOVA decoder also produces BER improvements as the number of iterations increase, although these are not as large as those obtained with Log-MAP and results are very similar, whether AWGN scaling is used, or not.

Figures 6.29 and 6.31 show the BER results for the combination of SRK with turbo codes. The Log-MAP decoder shows results similar to those experienced with the time-invariant channel,

performance is reduced as the number of iterations increases and, although the use of AWGN channel scaling produces better results after a single pass through a Log-MAP decoder, the introduction of iterative decoding causes much higher degradation than with no scaling factor at all. SOVA turbo decoding also produces degradation in performance with AWGN scaling. The difference between the SOVA decoding results and the Log-MAP results is that the omission of a scaling factor has little effect on the SOVA decoder.

Table 6.7 shows some points of interest in the BER simulation results, which help to compare the two decoding algorithms as well as the two equaliser update algorithms.

			Component Decoding Algorithm and Iteration Number					
Equaliser	Scaling Factor	BER	Log-MAP 1/2	SOVA 1/2	Log-MAP 1	SOVA 1	Log-MAP 8	SOVA 8
LMS	AWGN Reliability	10^{-2}	2.11dB	0.93dB	1.78dB	0.33dB	1.57dB	0.33dB
LMS	AWGN Reliability	10^{-4}	5.78dB	6.05dB	5.083dB	5.5dB	4.722dB	5.33dB
SRK	AWGN Reliability	10^{-2}	2dB	2.11dB	-	6.9dB	-	-
SRK	AWGN Reliability	10^{-4}	6.5dB	6.375dB	-	-	-	-
LMS	Lc = 1	10^{-2}	3.125dB	0.944dB	3dB	0.42dB	3dB	0.42dB
LMS	Lc = 1	10^{-4}	6.04dB	6.02dB	5.93dB	5.36dB	5.93dB	5.25dB
SRK	Lc = 1	10^{-2}	3.385dB	2.133dB	3.615dB	7.5dB	3.615dB	-
SRK	Lc = 1	10^{-4}	7.125dB	6.55dB	7.261dB	-	7.261dB	-

Table 6.7: Points of Interest from Bit Error Rate Results for Combination of DFE Equalisers with Turbo Codes over ETSI Indoor Channel

Considering the simulations that were made with AWGN channel scaling first, it is immediately obvious that the SOVA turbo decoder outperforms the Log-MAP decoder where the LMS equaliser update algorithm is used at a voice quality BER and, indeed, for virtually the whole of the simulated range. Here, the SOVA improvement at a BER of 10^{-2} was 1.18dB after one half iteration, 1.45dB after one turbo iteration and 1.24dB after eight iterations. The situation begins to become reversed at higher BERs, and, at the second point of interest, Log-MAP shows gains over SOVA of 0.27dB, 0.417dB and 0.608dB after the same number of iterations. The SNR at which SOVA ceases to outperform Log-MAP gets lower as the number of iterations increase. SOVA is the better performer until a BER of $5e^{-4}$ is reached at 5.75dB after one half-iteration. This reduces to $4.2e^{-4}$ at 5.2dB after one full iteration and $1.6e^{-3}$ at 4dB after eight iterations.

Where the SRK update algorithm was used, with AWGN channel scaling, the iterative decoding of the turbo decoder produces degradation in performance. Both decoders suffer under these circumstances, and in similar ways to those experienced for the same systems under the time-invariant channel conditions imposed earlier. The Log-MAP decoder degrades to such an extent that the BER results after multiple iterations actually increase with SNR, whereas the SOVA decoder results, when in combination with the SRK equaliser, despite still suffering intense distortion, retain a downward gradient. The results obtained for one half-iteration show that this combination produces better results for Log-MAP at lower BERs, with a gain over SOVA at 10^{-2} of 0.11dB. This is reversed at the higher point of interest, with SOVA returning a gain of 0.125dB over Log-MAP at 10^{-4}.

Considering the omission of any kind of channel scaling, the two decoding algorithms still produce improved results where the number of iterations is increased with the LMS equaliser, although the improvements from one iteration to eight iterations are reduced, much more obviously in the case of Log-MAP decoding. In fact, the performance of the Log-MAP turbo decoder after eight iterations is worse than that of a single Log-MAP decoder with AWGN scaling for the SNR range examined. It does appear, however, that the steeper gradient of the un-scaled decoder would overtake the scaled decoder at higher SNRs. The results for the SOVA decoder, using LMS, are very similar to those obtained with AWGN channel scaling, slightly better, in fact. This combination alters the gradient of the SOVA combination such that the voice quality BER is obtained at a higher SNR, whereas the data quality BER is reached at a lower SNR. The reduction in performance caused by this combination on the Log-MAP decoder shows a loss of around 1dB to 1.5dB at both points of interest for all iterations recorded, when compared to the results using AWGN channel scaling. All this means that the SOVA decoder shows gains of an even higher margin than those experienced with AWGN channel scaling. At a BER of 10^{-2}, the gains are 2.181dB after one half-iteration and 2.58dB after both one iteration and 8 iterations. Unlike the results obtained with AWGN scaling, SOVA continues to show a gain over its Log-MAP equivalent, with a gain of 0.02dB after one half-iteration, 0.57dB after one iteration and 0.68dB after eight iterations.

The SRK simulations with no scaling follow the same patterns that were found with the time-invariant channel. The Log-MAP decoder shows a small degradation from one half-iteration to eight iterations, whereas the SOVA decoder reacts in the same way as it did with the same equaliser and AWGN channel scaling, degrading as the number of iterations increases. The interesting point to note is that, of the half-iteration results, it is the SOVA decoder which is the better performer, at both high and low BERs. The gain for this decoder over Log-MAP at 10^{-2} is 1.252dB, reducing to 0.575dB at 10^{-4}. Of course, after one or more iterations, the Log-MAP decoder shows better results due to the fact that it is not severely affected by the lack of useful extrinsic information.

Of the two equaliser update algorithms in combination with turbo decoders, the LMS algorithm appears to be much less affected than the SRK algorithm by incorrect scaling, or lack thereof. Both decoders exhibited performance improvements with increasing numbers of iterations with this equaliser and, generally, the AWGN channel scaling produced equal, or better, performance. The SRK equaliser showed better results for half-iteration performance with AWGN channel scaling than

without, however, these were not as good as those obtained with the simpler LMS algorithm. The effect upon the turbo decoder with AWGN channel scaling and the SRK equaliser was catastrophic for both algorithms. Although the effect appears more obvious in the Log-MAP results, the performance degradation was higher with SOVA. The lack of channel scaling reduced performance for the Log-MAP decoder, but not to the extent that using AWGN scaling did. No channel scaling had little effect on the SOVA decoding algorithm, results were still poor for turbo decoding, showing a major reduction in performance compared to the Log-MAP, in a similar way to that encountered with AWGN scaling.

The frame error rate results follow the same patterns put forth by the BER results above. The only matter that needs addressing is the apparent reduction in FER performance in both Log-MAP decoders where channel scaling is omitted. It appears that the Log-MAP decoder frame error rates are more severely affected by this lack of scaling than the bit error rates and that the divergence from the results obtained with scaling is more pronounced than in the BER results. This brings about very low FER values for this decoder when combined with either equaliser using no channel scaling.

6.6.7 16-state Turbo Code Pedestrian Channel Simulations

The same decoder/equaliser combinations described in 6.6.5 were then subjected to the ETSI pedestrian mobile channel model. This channel takes into account Doppler shift at a walking speed of 4mph (roughly 6.5kph) and is described, in greater detail, in chapter 3.

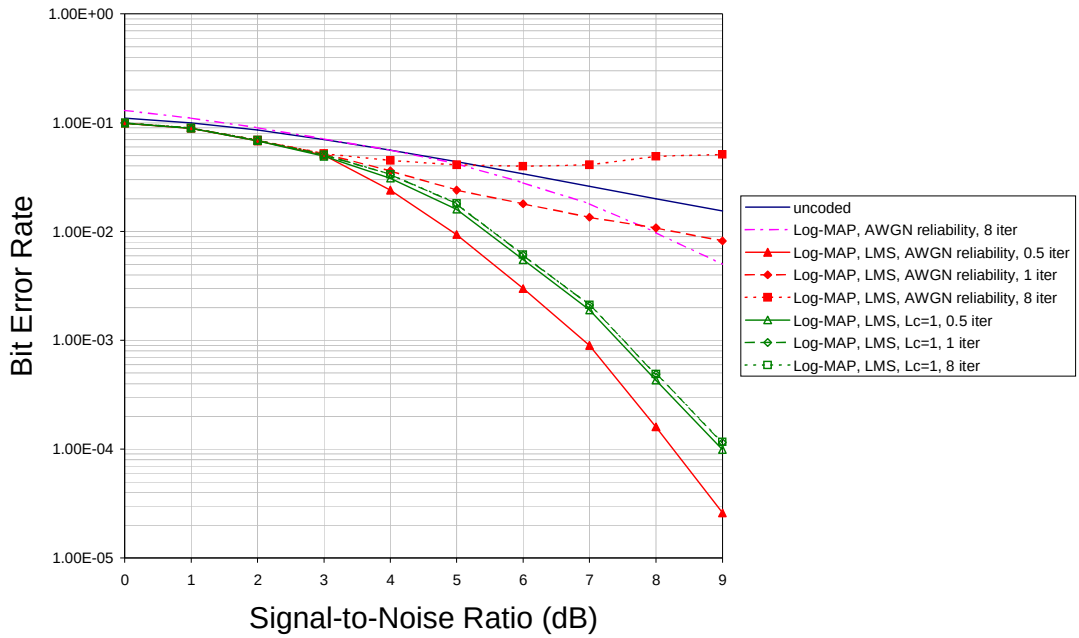


Figure 6.33: BER for Log-MAP Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Pedestrian Channel

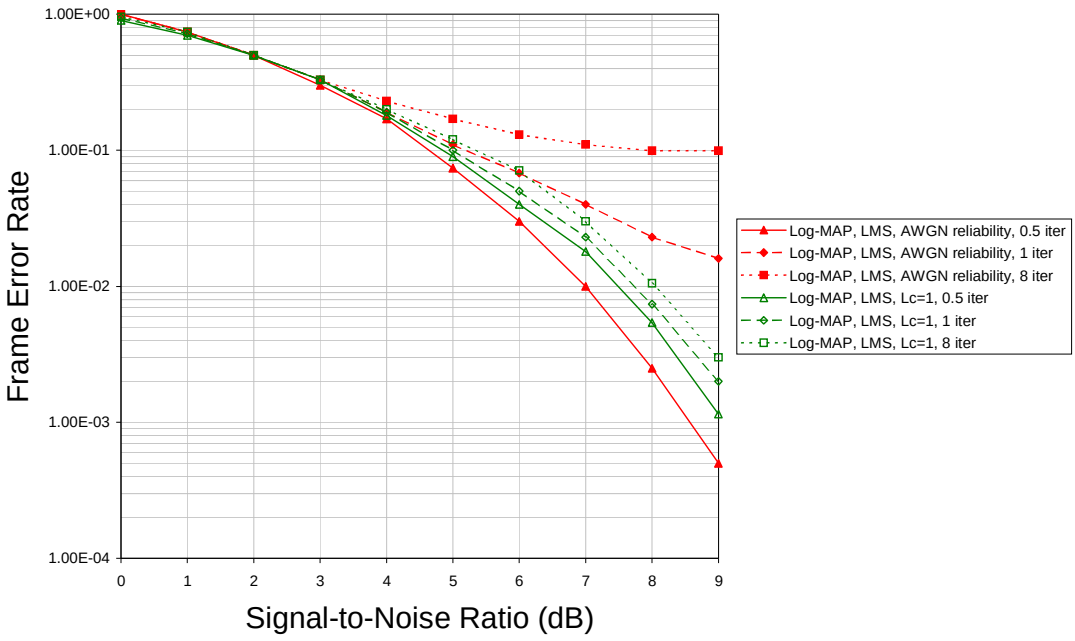


Figure 6.34: FER for Log-MAP Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Pedestrian Channel

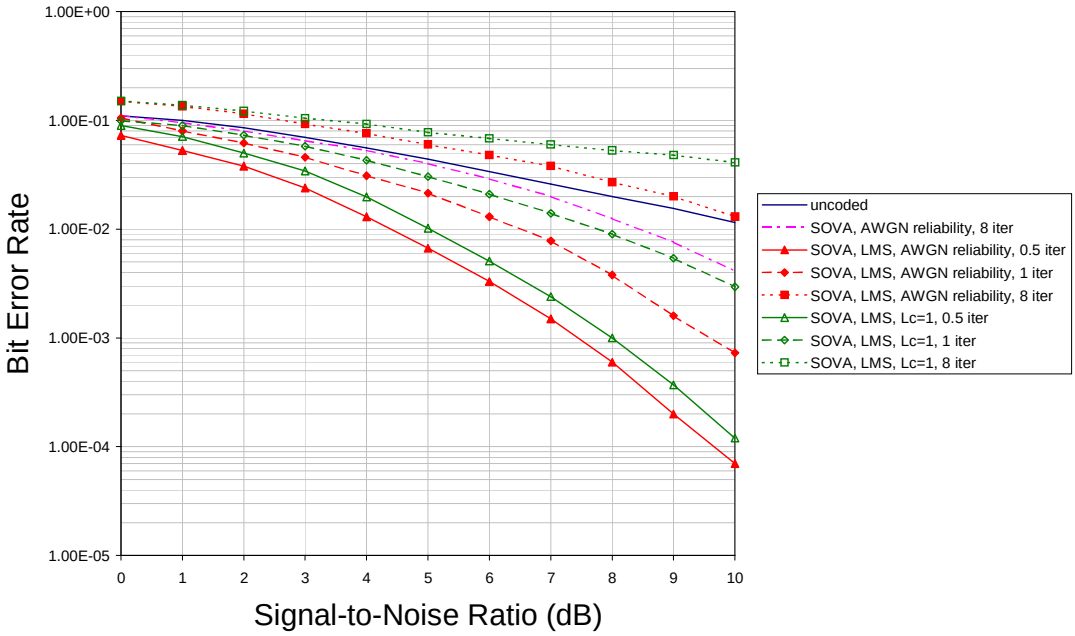


Figure 6.35: BER for SOVA Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Pedestrian Channel

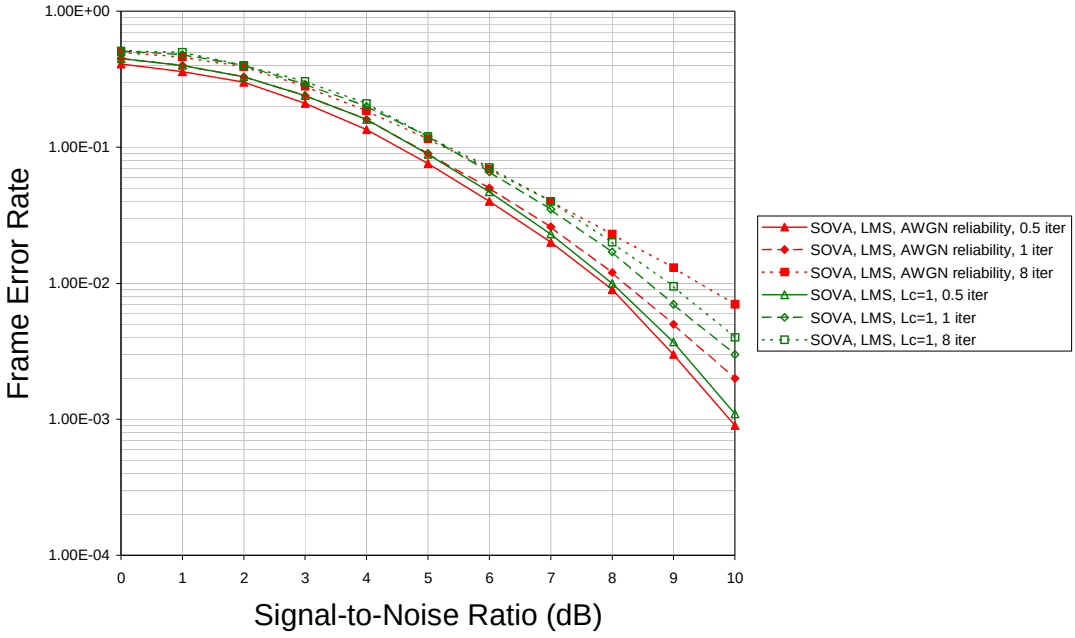


Figure 6.36: FER for SOVA Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Pedestrian Channel

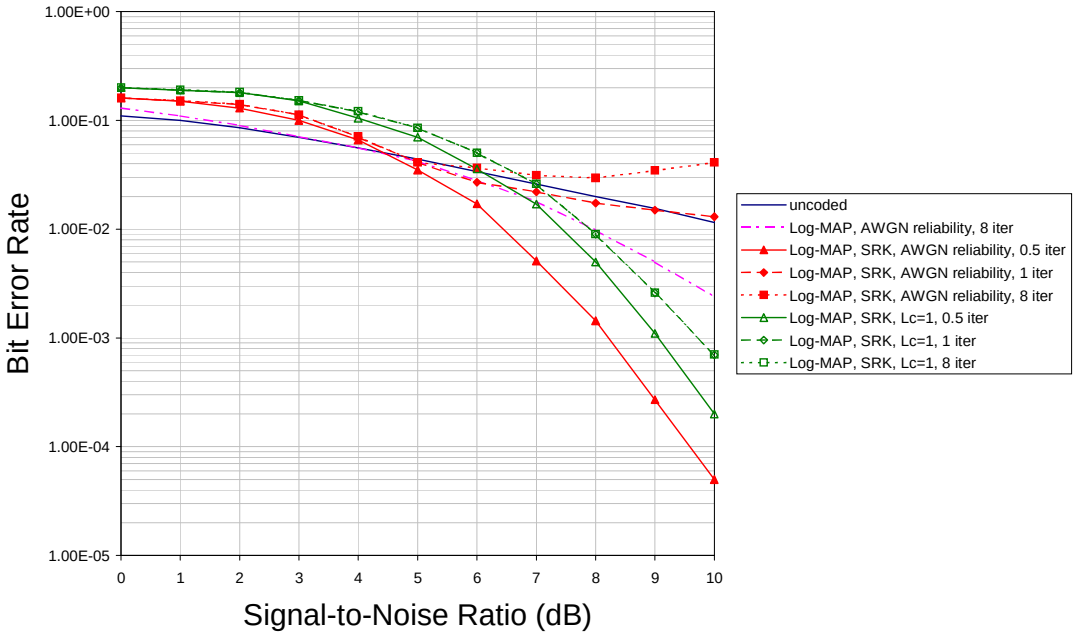


Figure 6.37: BER for Log-MAP Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Pedestrian Channel

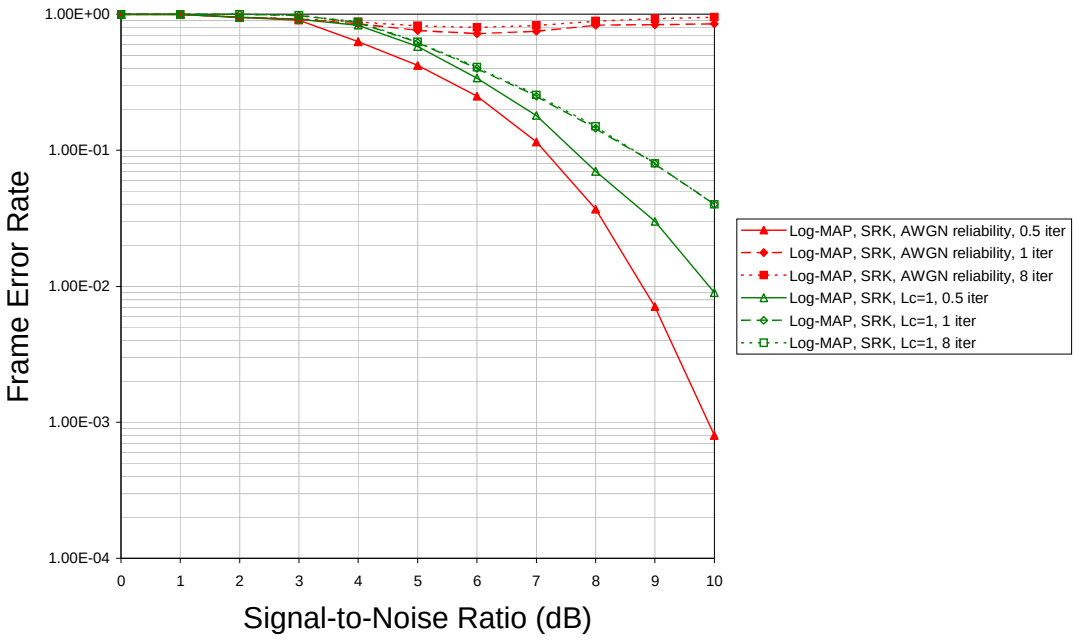


Figure 6.38: FER for Log-MAP Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Pedestrian Channel

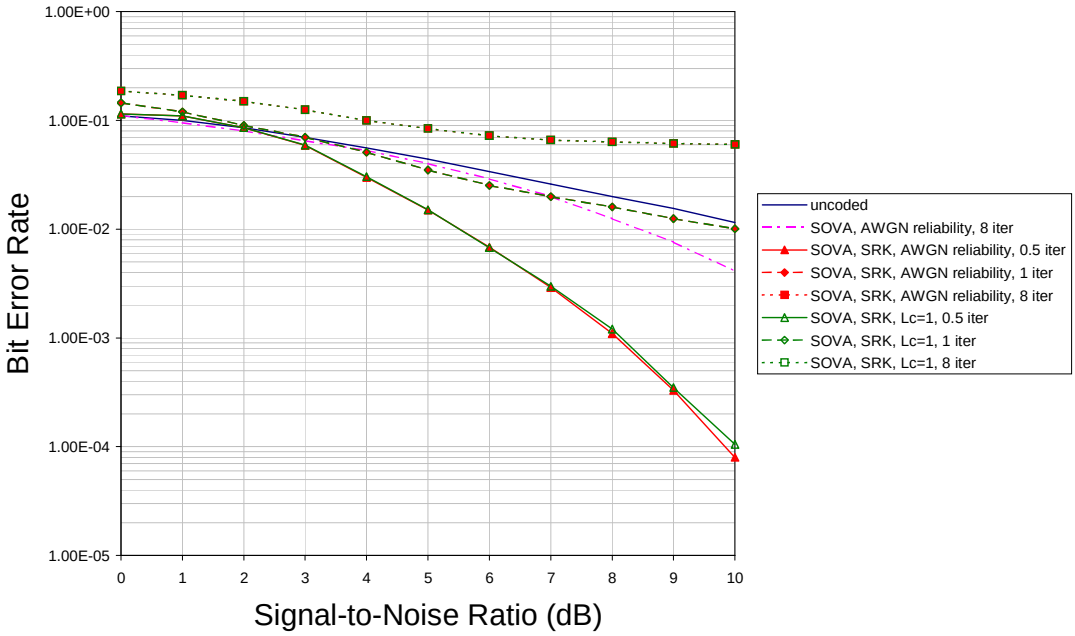


Figure 6.39: BER for SOVA Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Pedestrian Channel

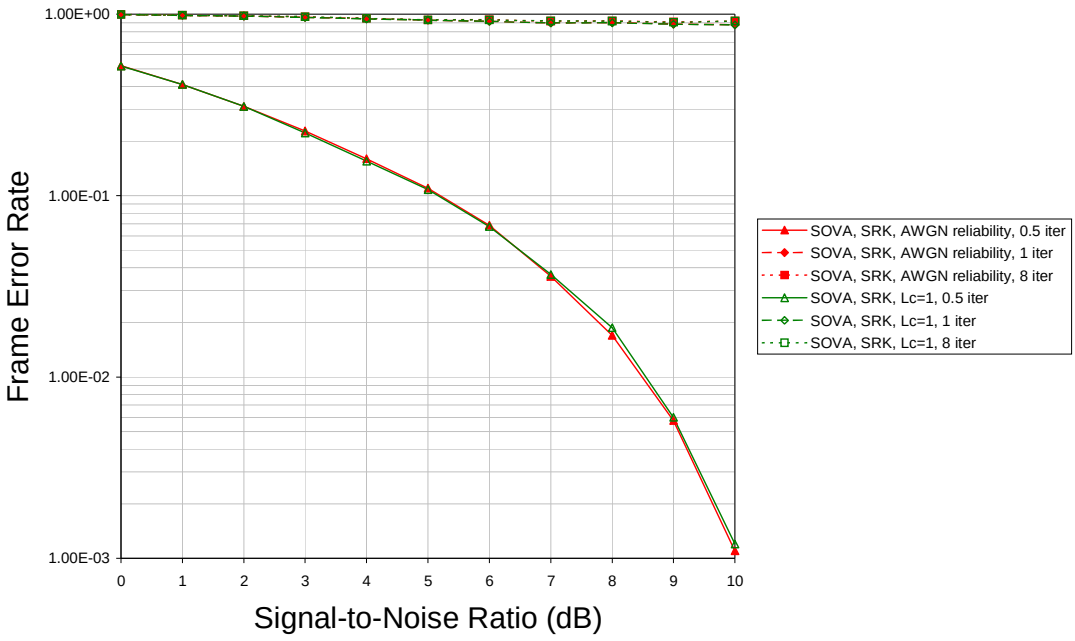


Figure 6.40: FER for SOVA Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Pedestrian Channel

6.6.8 Discussion of Pedestrian Channel Simulation Results

It is interesting to note, from the figures above, that the bit error rate performance curves are beginning to resemble the static results obtained earlier in this chapter, and move away from the results experienced when turbo codes are applied to AWGN channels. The effect of the increased ISI and Doppler can readily be seen when comparing the uncoded results of the indoor and pedestrian results. No set of results in the pedestrian simulations shows the improvement produced by iterative decoding that was evident in some of the indoor channel simulations. This indicates that the turbo code/DFE equaliser combination only works in circumstances where ISI is minimal. Note also that, as with the static results, SOVA retains the downward curve, while Log-MAP suffers to such an extent that results begin to get worse as SNR increases. Table 6.8 highlights some points of interest for further investigations into the results obtained over pedestrian channels.

			Component Decoding Algorithm and Iteration Number					
Equaliser	Scaling Factor	BER	Log-MAP 1/2	SOVA 1/2	Log-MAP 1	SOVA 1	Log-MAP 8	SOVA 8
LMS	AWGN Reliability	10^{-2}	4.83dB	4.4dB	8.25dB	6.45dB	-	-
LMS	AWGN Reliability	10^{-4}	8.25dB	9.66	-	-	-	-
SRK	AWGN Reliability	10^{-2}	6.43dB	5.5dB	-	10dB	-	-
SRK	AWGN Reliability	10^{-4}	9.6dB	9.86dB	-	-	-	-
LMS	Lc = 1	10^{-2}	5.43dB	5dB	5.52dB	7.71dB	5.52dB	-
LMS	Lc = 1	10^{-4}	9dB	10.17dB	9.11dB	-	9.11dB	-
SRK	Lc = 1	10^{-2}	7.43dB	5.5dB	7.86dB	10dB	7.86dB	-
SRK	Lc = 1	10^{-4}	10.4dB	10dB	-	-	-	-

Table 6.8: Points of Interest from Bit Error Rate Results for Combination of DFE Equalisers with Turbo Codes over ETSI Pedestrian Channel

As with the indoor channel discussion, the simulations performed with AWGN channel scaling are considered first.

From table 6.8, it can be seen that when combined with LMS, the SOVA decoding algorithm again returns better results than the Log-MAP version. As stated earlier, the use of iterative decoding degrades the decoder/equaliser performance whichever decoding algorithm is used, a reversal of the results encountered under indoor channel conditions. This does not stop the SOVA decoder producing a gain at voice quality BERs after one half-iteration of 0.43dB, which is increased to 1.8dB by the end of the first full turbo iteration. The Log-MAP combination results suffer much worse degradation than the SOVA decoder in this combination, which is the reason for the large decoder gain increase after one iteration. In a similar way to many other results seen here, the Log-MAP decoder shows an improvement on the SOVA results at data quality BERs with a gain of 1.41dB after one half-iteration.

The SRK equaliser when combined with the turbo decoder, using AWGN channel scaling, produces similar curves for both decoders. Degradation is similar to that encountered in the time-invariant simulations, causing performance to reduce with increasing numbers of iterations and increasing SNR. As with the LMS results, SOVA produces better results at voice quality BERs, with a gain of 0.93dB. SOVA also attains 10^{-2} after one full turbo iteration, unfortunately, the loss is some 4.5dB, but this does illustrate the fact that the SOVA decoder at least retains a downward bent. Again, as before, the Log-MAP decoder produces better performance at lower BERs with a gain of 0.26dB at 10^{-4}.

Where results are comparable, it can be seen that the LMS algorithm reaches both voice quality and data quality BERs with either decoder, using AWGN channel scaling, at a lower SNR than SRK. The LMS shows gains after one half-iteration of 1.6dB for Log-MAP and 1.1dB for SOVA at 10^{-2} and 1.35dB and 0.2dB for the same decoders at 10^{-4}.

Looking at the four combinations with no channel scaling, differences in the way the algorithms react begin to emerge. To begin with, the LMS algorithm does not cause a major degradation in the results produced by the Log-MAP decoder, although the results after one half-iteration are not as good as with AWGN channel scaling. The reaction of the SOVA decoder with LMS and no scaling is similar to that with AWGN scaling, but somewhat worse. Table 6.8 shows this. The Log-MAP decoder, when used for one half-iteration, reaches voice quality BERs 0.6dB later with no scaling and reaches data quality BERs 0.75dB later. However, the lack of catastrophic degradation means that, even after 8 iterations, the un-scaled Log-MAP/LMS equaliser combination shows a result that is similar to that produced after one half-iteration. The combination of SOVA with LMS and no scaling produces similar performance to those found with AWGN scaling. The lack of scaling reduces the effectiveness of the decoder but results follow those obtained with scaling. The loss incurred is 0.6dB at 10^{-2} after one half-iteration and 1.26dB after one iteration. At data quality BERs, the loss in performance is slightly reduced to 0.51dB after one-half iteration. Of the two combinations using LMS and no scaling, SOVA produces better results at voice quality BERs, showing a gain of 0.43dB at 10^{-2}, but Log-MAP produces better results at lower BERs, with a gain of 1.17dB at 10^{-4}. These results are after one-half iteration, after which the SOVA decoder begins to suffer degradation making comparison meaningless.

Examining the SRK combinations with no scaling, it can be seen that the Log-MAP decoder again reacts differently to when scaling is included. The SOVA, on the other hand, acts in a very similar way. The Log-MAP decoder with SRK reacts in a similar way to the same decoder with LMS, the degradation caused by iterative decoding is present, but is not catastrophic. Performance is reduced from results where scaling was included, with a loss after one half-iteration of 1dB at voice quality BERs and 0.8dB at data quality BERs. As with the LMS equaliser under these circumstances, degradation is experienced but does not increase at the same rate as AWGN scaling. The SOVA decoder in combination with SRK and no channel scaling produces results that are again very close to those obtained using scaling. In fact, in this instance, there is no difference at voice quality BERs. A slight difference does begin to appear at lower BERs, with a loss of 0.14dB at 10^{-4}. As with the Log-MAP results, these are obtained after one half-iteration as degradation influences the later results.

The FER results follow the BER results. The SOVA decoder FER results appear to be much closer, where scaling and no scaling are compared, whereas the Log-MAP decoder combinations that exhibited degradation in BER appear to show this more in the FER results.

6.6.9 16-state Turbo Code Vehicular Channel Simulations

Finally, results were obtained for the same decoder/equaliser combinations over the ETSI vehicular channel model. In this case, the channel simulates movement at 70mph (113kph). Further explanation of this channel is again found in chapter 3.

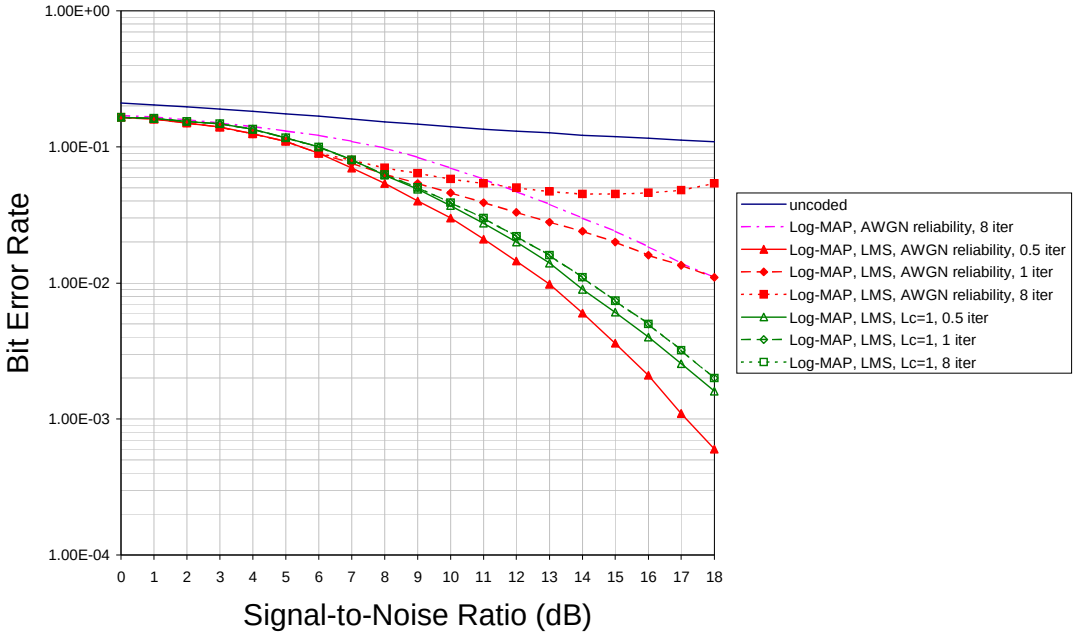


Figure 7.41: BER for Log-MAP Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Vehicular Channel

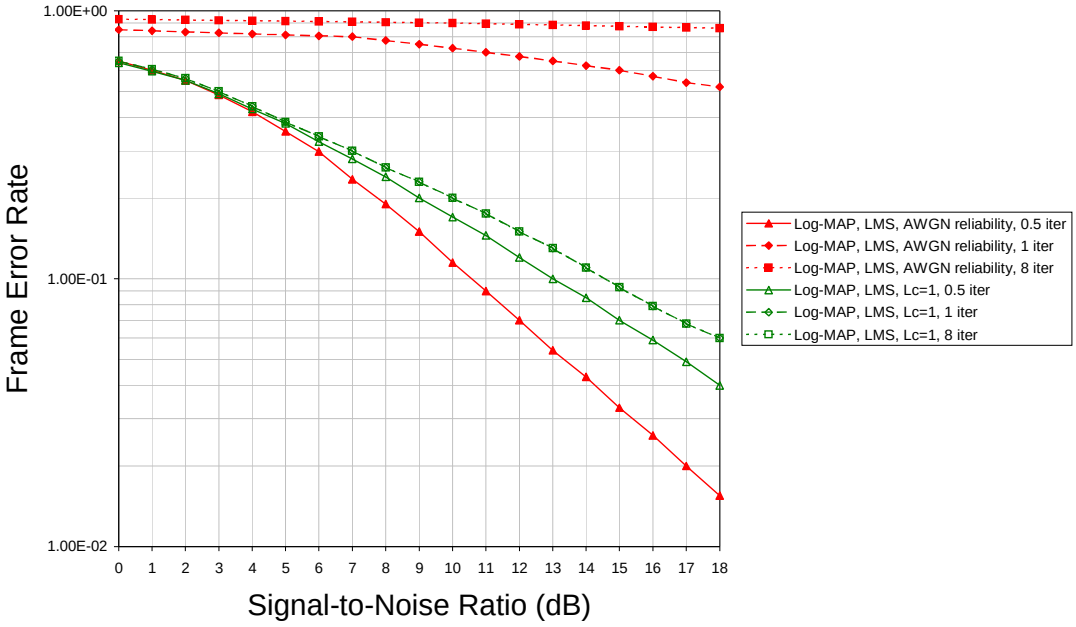


Figure 6.42: FER for Log-MAP Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Vehicular Channel

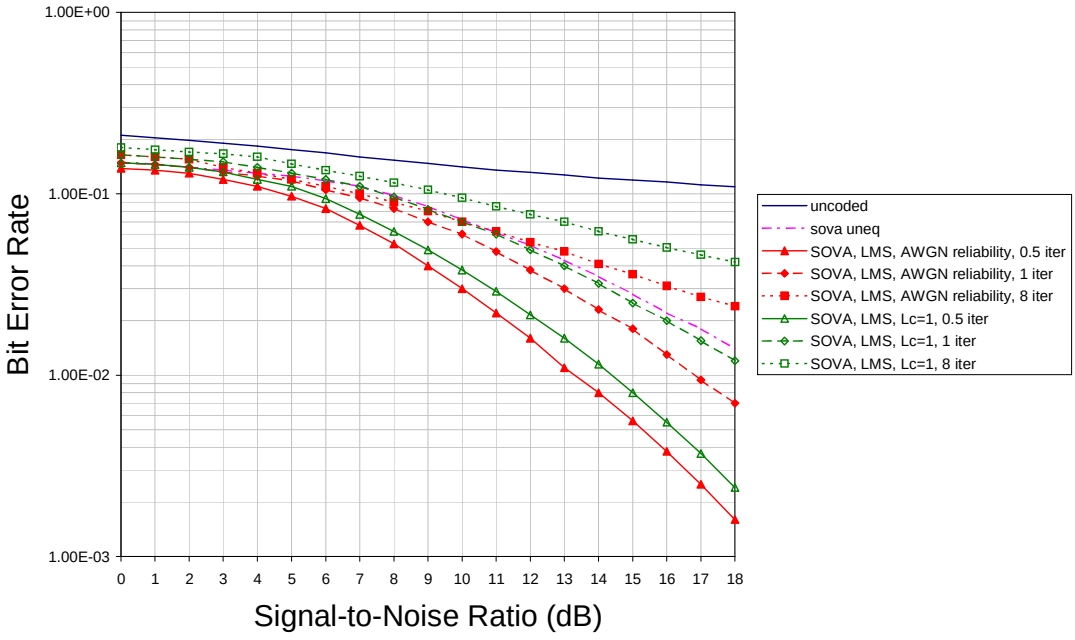


Figure 6.43: BER for SOVA Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Vehicular Channel

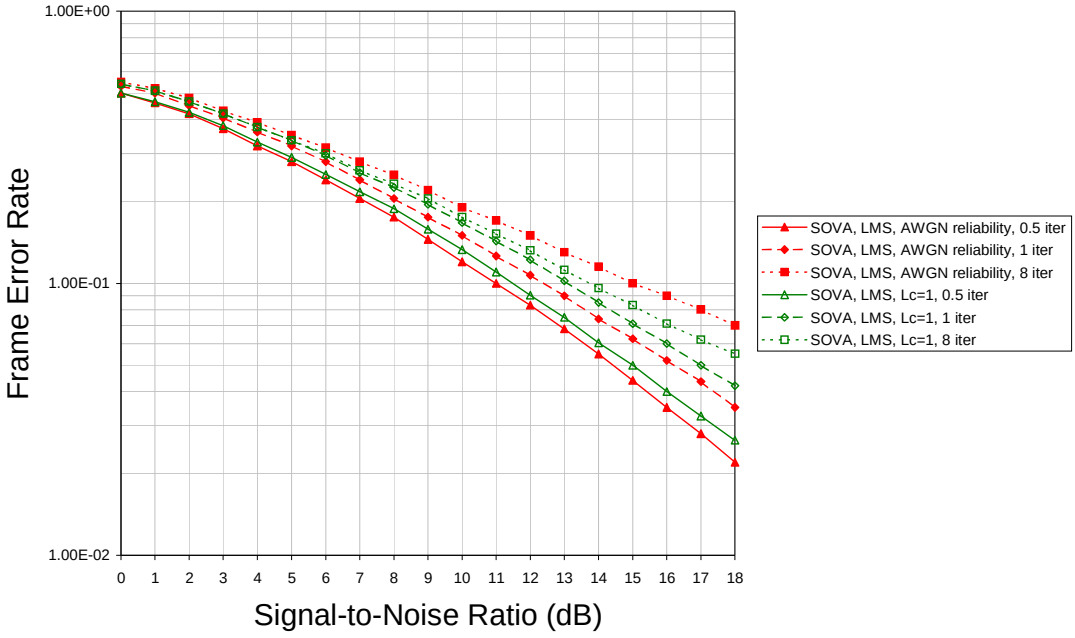


Figure 6.44: FER for SOVA Turbo Decoding Combined with DFE Equalisation using LMS Update Algorithm over ETSI Vehicular Channel

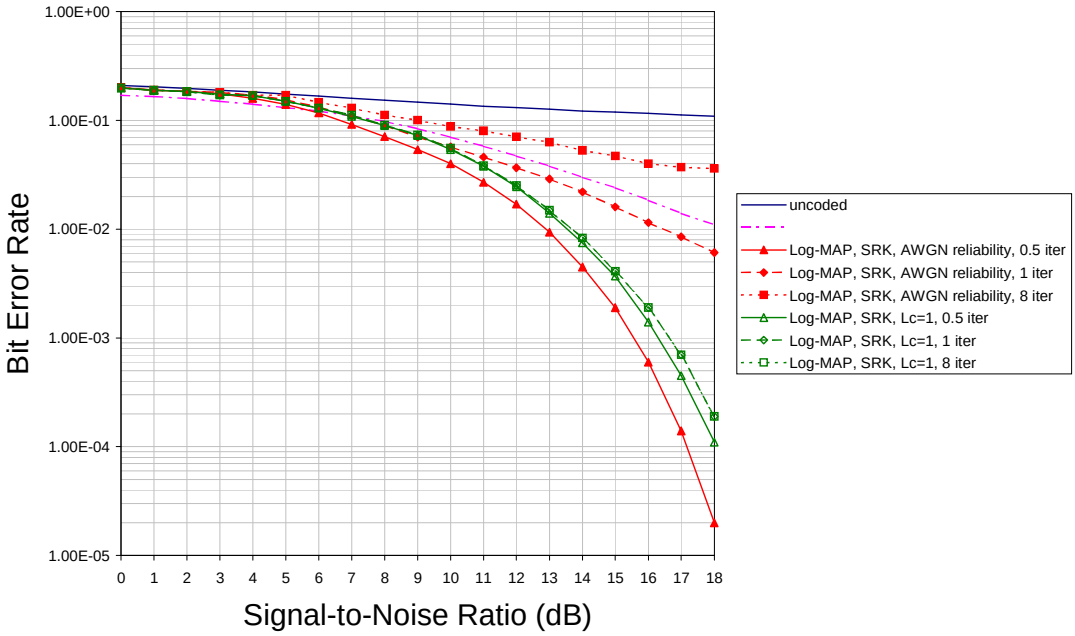


Figure 6.45: BER for Log-MAP Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Vehicular Channel

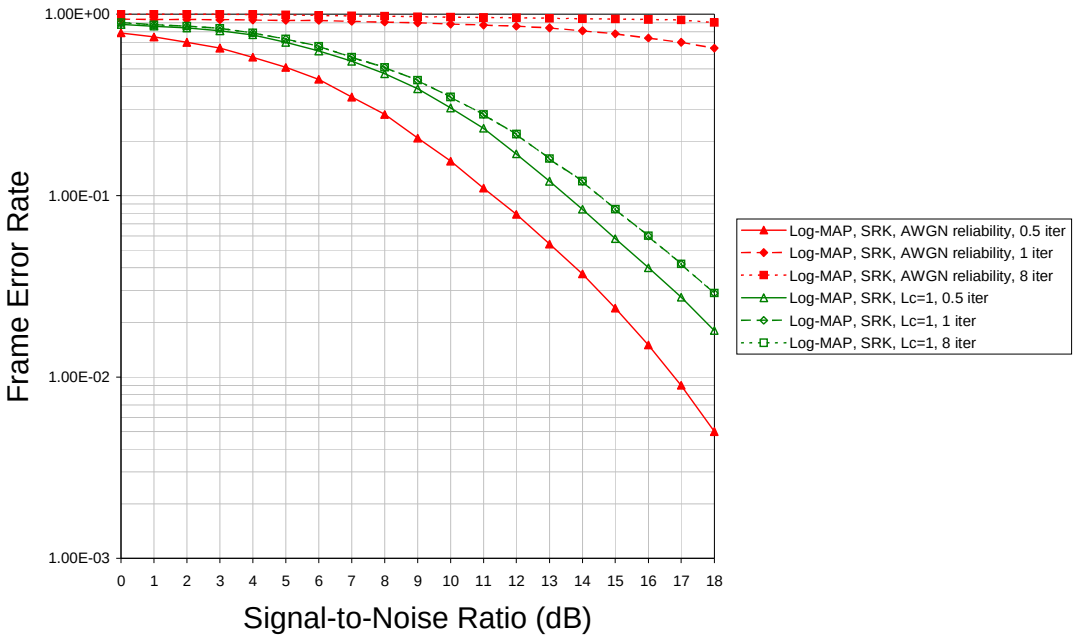


Figure 6.46: FER for Log-MAP Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Vehicular Channel

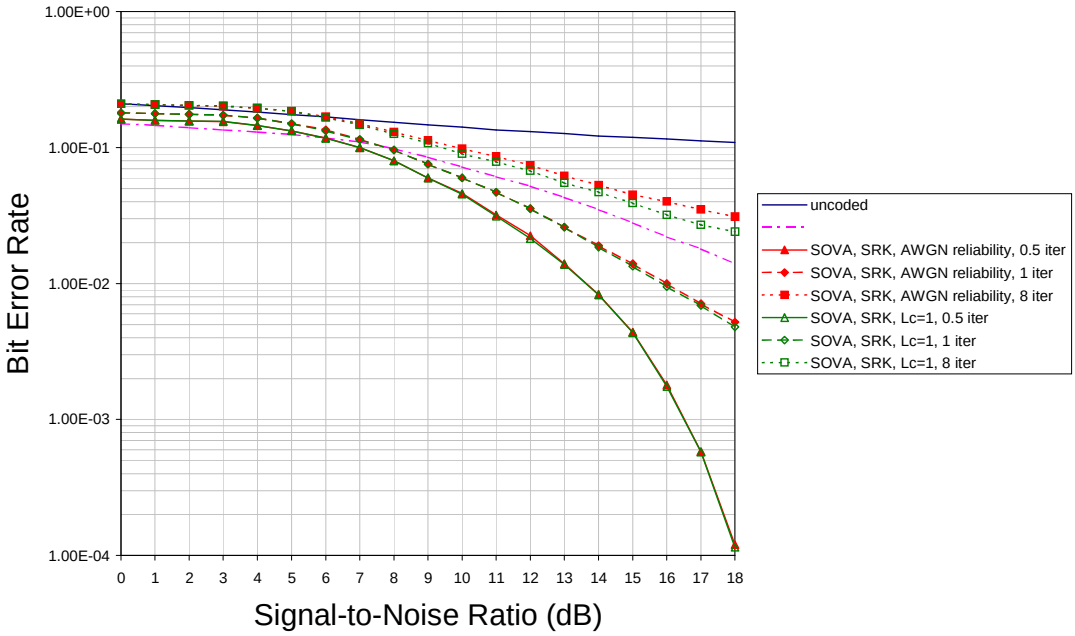


Figure 6.47: BER for SOVA Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Vehicular Channel

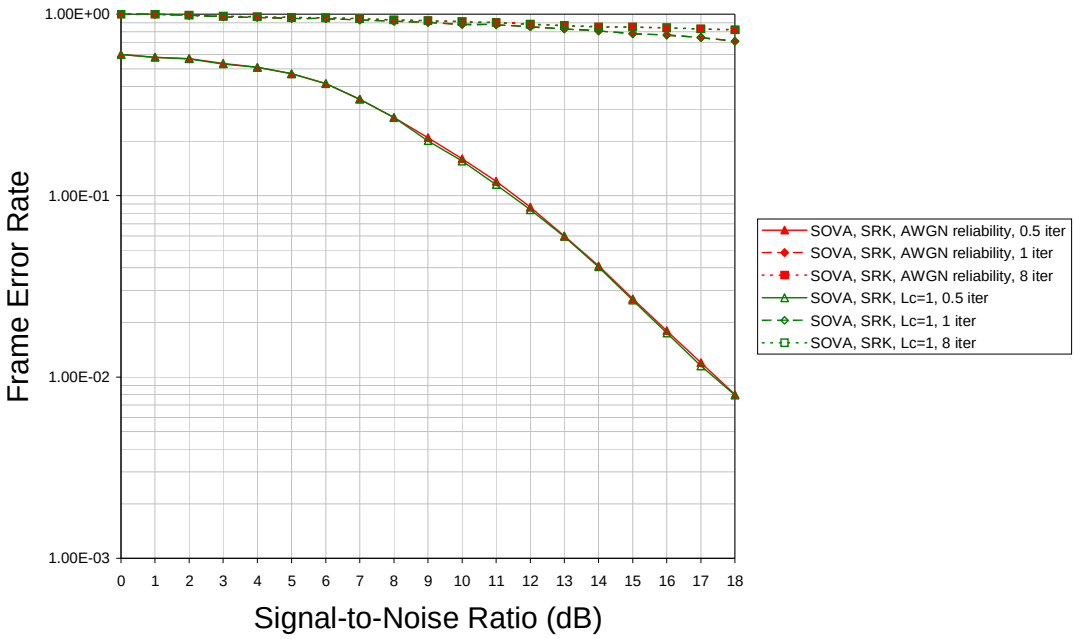


Figure 6.48: FER for SOVA Turbo Decoding Combined with DFE Equalisation using SRK Update Algorithm over ETSI Vehicular Channel

6.6.10 Discussion of Vehicular Channel Simulation Results

These results appear to follow similar patterns to the pedestrian simulations. Notice that the combinations that were successful in earlier simulations are less so under these conditions. The SOVA decoder does not outperform Log-MAP so obviously at higher BERs and the LMS algorithm does not exhibit as large a gain over its more complex counterpart. Also notice that these simulations do not reach the same lower BERs as other simulations. Table 6.9 highlights the points of interests for further discussion.

			Component Decoding Algorithm and Iteration Number					
Equaliser	Scaling Factor	BER	Log-MAP 1/2	SOVA 1/2	Log-MAP 1	SOVA 1	Log-MAP 8	SOVA 8
LMS	AWGN Reliability	10^{-2}	13dB	13.3dB	18.2dB	16.83dB	-	-
LMS	AWGN Reliability	10^{-3}	17.14dB	-	-	-	-	-
SRK	AWGN Reliability	10^{-2}	12.9dB	13.6dB	16.2dB	16dB	-	-
SRK	AWGN Reliability	10^{-3}	15.5dB	16.5dB	-	-	-	-
LMS	Lc = 1	10^{-2}	13.8dB	14.4dB	14.25dB	-	14.25dB	-
LMS	Lc = 1	10^{-3}	-	-	-	-	-	-
SRK	Lc = 1	10^{-2}	13.5dB	13.6dB	13.66dB	15.85dB	13.6dB	-
SRK	Lc = 1	10^{-3}	16.29dB	16.5dB	16.66dB	-	16.66dB	-

Table 6.9: Points of Interest from Bit Error Rate Results for Combination of DFE Equalisers with Turbo Codes over ETSI Vehicular Channel

Beginning with the combinations using AWGN channel scaling, it can be seen that the Log-MAP combination shows gains at both highlighted BERs. The shallow nature of the SOVA decoder in these results means that even a BER of 10^{-3} is not reached, although the results indicate a loss of around 2.5dB compared with the Log-MAP/LMS combination at this point, a significant increase on the 0.3dB loss at a voice quality BER of 10^{-2}. These results occur after one half-iteration, but the degradation suffered by the Log-MAP combination is such that the SOVA results are better after one whole iteration by 1.37dB. Figures 6.41 and 6.43 show the difference between the degradation suffered by the two combinations and it is clear that the effect of incorrect extrinsic information affects Log-MAP more than SOVA with BERs obtained actually increasing with SNR for Log-MAP after 8 iterations.

Looking at the two combinations using AWGN scaling and SRK equalisation, the Log-Map combination shows gains at both points of interest, increasing slightly with SNR from 0.7dB at a BER of 10^{-2} to 1dB at 10^{-3}. These two combinations also reach data quality BERs of 10^{-4}, lower points than the equivalent LMS results, with log-MAP showing a gain of 1.1dB over SOVA here. Again, the losses experienced with increased number of iterations are larger in the Log-MAP results, with a loss of 3.3dB for Log-MAP at 10^{-2} compared with 2.4dB for SOVA when increasing from one half-iteration to one whole iteration.

Where no channel scaling values were used, the combinations react in very similar ways to those seen in the results obtained for the pedestrian channel. Considering the LMS combinations first, the Log-MAP combination shows little reduction in performance from one iteration to the next and also

shows a gain in performance compared to SOVA of 0.6dB at 10^{-2}. Note that neither combination reaches the second point of interest, but that the Log-MAP combination suffers such small losses that after eight iterations, the results still show a gain over the half-iteration SOVA results of 0.15dB at 10^{-2}.

The SRK combinations with no channel scaling react in the same way as the LMS combinations under these circumstances. The Log-MAP combination looses a small amount of performance with increasing iterations, whereas the SOVA decoder shows increasing losses with increasing number of iterations. The two combinations reach voice quality BERs at similar SNRs after one half-iteration, with the Log-MAP combination showing a small gain of 0.1dB over SOVA. This gain increases to 0.21dB at the second point of interest. The losses experienced by the two combinations in later iterations can be illustrated by looking at the gain at 10^{-2}. At this point, after one full iteration, the decoder gain for the Log-MAP combination is 2.19dB.

Comparing the two equaliser algorithms, it is immediately seen that, for the half-iteration results, the SRK equalisers when combined with Log-MAP produce better results, at both points of interest and for all numbers of iterations. The case is almost the same for the SOVA combinations. The SRK algorithm produces better results than LMS when combined with SOVA, except where AWGN channel scaling is used and voice quality BERs are required with one half-iteration.

Comparing the effects of the two channel scaling methods, it is still evident that AWGN scaling values produce better results after one half-iteration, but degradation with these values when the number of iterations is increased means that, in some cases, omitting scaling altogether produces better results for turbo codes.

As before, the frame error rate curves follow the same patterns as the bit error rate curves, although from the graphs the degradation that appears when increasing the number of iterations causes greater reductions in performance.

6.7 Conclusion

Inter Symbol Interference encountered during multipath propagation in a communications scheme can have catastrophic effects on the error performance of the system. Error control codes go some way in combating these perturbations but it is sometimes necessary to combine them with other devices to improve their chances of succeeding.

This chapter gives a theoretical background to the equalisation techniques in this report. Beginning with a definition of adaptive equalisation and an explanation of its role in the communications system, the chapter then went on to analyse some of the many schemes that have been developed over the years.

The most basic equaliser system, the linear transversal equaliser, was first discussed with its definition and applications.

This was followed by a more thorough discussion of the decision feedback equaliser, a more effective, but more complicated, system. Two of the most popular updating schemes for this type of equaliser were then defined and mathematical derivations examined.

The first system discussed was the least mean squared algorithm, a simple and quick update algorithm that is still widely used. The updating method was described together with an explanation of the ways in which the system must be initialised and tuned during operation.

The second method described was the recursive least squares algorithm. The system was fully derived, including derivations of some of the mathematical tools necessary to complete the derivation. This was accompanied by a discussion of the initialisation of the system.

The method used to combine what is conventionally a hard decision equalisation device with a turbo decoder was then explained.

The idea behind using such a device was that the complexity of the system would be kept to a minimum and, therefore, the decoding time could be kept to within acceptable limits for applications that are likely to use short frame transmission systems.

Until now, all research in the area of ISI with respect to the turbo effect has been conducted using turbo equalisation, a process that replaces one SISO component decoder with a SISO equaliser. The process is more complex than the turbo decoder system due to the re-arranging of data passed between the two devices (relocation of pilot symbols each iteration, channel interleaving etc.). As a result of being more complex, the decoding delay is increased.

It was acknowledged that the combination of the DFE equaliser and turbo decoder would not allow proper soft-decision inputs to the turbo decoder. However, the intention was that this might be overcome through channel scaling.

To this end, two systems were implemented. The first used AWGN channel reliability values, as the equaliser should have pre-processed the received data and reduced the effects of ISI. The second system used a channel reliability of one. This effectively omitted the channel reliability scaling factor, as it was noted that the output of the equaliser, prior to the decision device, still appeared to be binary information once the equaliser had stabilised.

These two systems were implemented with both the previously investigated decoding algorithms and using two of the most popular equaliser update algorithms, LMS and SRK, to process the received data prior to decoding.

The combinations were tested over various ISI channels, ranging from dynamic indoor, with an uncoded curve similar to AWGN, to dynamic vehicular and static channels, with very harsh uncoded curves.

It was shown that, without equalisation, turbo codes produced only a small improvement over uncoded information when transmitted over all channels considered in this chapter and that, where investigated, equalisers without codes produced better results.

The results from the equaliser and turbo decoder combinations were an improvement on turbo codes alone, but were still poor when compared to uncoded, equalised, results.

In fact, improvements on uncoded, equalised, results were only obtained when the combinations were examined after one half iteration of the decoders. After this, performance was severely reduced. It was found that, no matter which combination of scaling factor, equaliser or decoder was used, performance was degraded as the number of iterations increased in all but the mildest cases of ISI. The indoor office environment, although dynamic, showed only a small reduction in uncoded performance to that of AWGN and, as such, the combination of the LMS algorithm with turbo codes produced improvements in performance as the number of decodes increased. All other channels investigated showed a reduction in performance as did the combination of SRK with turbo codes in the indoor environment. It appears that this effect is due to errors in the extrinsic information passed between component decoders, although it had only a very small effect in simulations with Log-MAP decoding and no channel scaling for some reason.

With regards to decoder performance in these combinations, the effects seen in previous simulations held for the most part. That is to say that the SOVA decoder tended to outperform Log-MAP at high, voice quality, BERs. The most notable exception was in the ETSI vehicular simulations, the most difficult situation examined, where Log-MAP proved to be the stronger performer at all points of interest, as was the case for all data quality BERs reached in other environments.

It was also the case that the simpler equaliser update algorithm was generally the better performer at higher BERs, all conditions except for the vehicular ISI showed a definite gain with this algorithm. In the vehicular channel simulations, LMS was only the better performer in combination with SOVA and AWGN scaling. At lower BERs, the situation was not so simple. For the indoor and pedestrian simulations, LMS still produced the better results, but for the two harshest environments SRK was generally the better algorithm in terms of lowest SNR required.

Different component codes were only examined on the time-invariant channel. A rather harsh channel, this showed that it is not always the case, as had previously been surmised, that codes with lower minimum effective free distance produced better performance at low SNRs. At almost all points of interest, it was the more complex component code that produced better results.

In conclusion, it cannot be said that these combinations were particularly successful in combating ISI and producing the kind of results that have come to be expected of turbo codes, except in mild cases like those experienced in office conditions. However, it is possible to use soft decoding techniques in combination with these equalisers to overcome the effects of even the harshest of ISI channels. The problem for turbo codes exists with the channel scaling, requiring the estimation of the fading amplitude upon each transmitted bit, something that cannot be accurately determined in channels with high levels of ISI.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

The aim of this thesis was to investigate short frame turbo codes and their reactions to different types of channel perturbation. Due to the applications in which these codes would most likely be used, the complexity, and therefore the induced delay, of the receiver systems were considered an important factor. Therefore, each strategy was developed with this in mind. The combination of decision feedback equalisers with turbo codes were also developed and investigated in an effort to counteract the effects of inter-symbol-interference while maintaining low complexity within the system.

Work began by reviewing the areas of turbo codes that had already been investigated and examining the conclusions that had been drawn about these codes and their abilities, as well as the effects of altering components within the scheme. Log-MAP decoding was found to be the preferred decoding scheme, due to its relatively low complexity and good bit error performance. It was also found that turbo codes, although considered, were not used in contemporary short frame systems because the increase in complexity over more conventional codes did not justify the improvement in BER performance that they provided. It was also found that little published work existed for turbo codes with short frames specifically and that many conclusions drawn for longer codes were taken as granted for shorter codes.

Although much less complex, the SOVA decoding scheme had been largely ignored in more recent years, due to its lower performance capabilities. This decision seemed sensible for long frame turbo code schemes as ultimate performance was required with less attention paid to decoder delay. However, it was seen that short frame schemes required a lower level of performance with much more attention paid to decoder delay.

Turbo codes were then examined in detail, showing how the turbo code scheme was a clever arrangement of previously researched devices. The use of a subset of convolutional codes, largely

ignored prior to turbo codes, was shown to be the basis for this scheme, largely because it required datawords with weights larger than one to produce codewords that diverge and later converge in the all-zero state. The combination of these devices with interleaving meant that codewords with almost infinite weight could be produced, an important step to reaching Shannon's theoretical channel capacity limit. This combination and the use of multiple decoders at the receiver, able to pass useful information to one another created a very powerful scheme.

Mathematical models for the decoders were given along with explanations of the various other components in the system. Methods used to estimate the qualities of these components and how to improve chosen schemes were also covered.

Following this, the theory of mobile communications channels was explained, along with the method used to modulate the digital information for transmission. One of the most basic channel models, AWGN, was considered, followed by Rayleigh fading and the use of tap-delay line channels to simulate Inter-Symbol Interference. The design of these channels using measured results as stipulated by UMTS, with a Doppler spectrum that is more realistic than those commonly used, was given for different environmental conditions.

Work then began on examining the performance of short frame turbo codes, to determine what differences, if any, existed between them and longer versions. Four codes were considered, with increasing constraint length. The first three codes were optimised or their size, while the largest code was not. Interleavers were designed using published methods to obtain better than average performance. The four codes were implemented with both the Log-MAP algorithm and the SOVA version and were compared for bit error and frame error performance. The effects of puncturing and frame length were first examined over AWGN channels. These results showed that the omission of certain parity bits reduced the code performance, as expected. SOVA was found to be the better performer at very low signal-to-noise ratios where puncturing was used, although Log-MAP soon began to outperform it. The rate at which the gain produced by the Log-MAP decoder over its SOVA counterpart increased was found to reduce as the effective free distance of the codes increased. It was also found that codes with smaller, but optimal, effective free distances produced better results than those with larger, non-optimal, effective free distances. It was also noted that the decoder gains for Log-MAP were larger for un-punctured codes than punctured, suggesting that SOVA was less affected by this loss of information.

The same codes were then considered with longer frames, still at least half the size of those commonly investigated. These simulations largely confirmed the earlier conclusions, that smaller optimal effective free distances produced better results at low SNRs, with the opposite being true at higher SNRs. It was also found that SOVA generally reached voice quality BERs before Log-MAP.

Simulations of the shorter, 100 bit dataword, codes were then made over Rayleigh fading channels, to examine the effect of errors in channel scaling due to a lack of channel amplitude at the decoder. It was found that the lack of correct information was not catastrophic, but that it did reduce the effectiveness of both decoding schemes. The effect was more obvious in the SOVA results, suggesting

that amplitude values were more important for this decoding scheme. In simulations where fading amplitude was known, Log-MAP was the better performer.

Having determined that SOVA was very close to, or better, than Log-MAP at higher BERs with short frames, investigations turned to the complexity of the two decoding schemes. It was found that Log-MAP was at least twice as complex as SOVA in number of operations. Simulations were therefore made on an equal number of operations basis. First, different numbers of decoder iterations were considered and SOVA was found to reach voice quality BERs at lower signal-to-noise ratios with equal, or less, complexity to Log-MAP. This did not continue through to higher SNRs however. It was also found that choosing a more complex component code, with SOVA decoding, of equal complexity to a less complex code and Log-MAP decoding could also, in some circumstances, produce better results at low BERs.

During these investigations, in comparison with published work, it was found that no investigations had been made into the effect of errors in SNR estimation at the SOVA turbo decoder, probably due to the fact that researchers preferred the Log-MAP method. Simulations were then made to determine the effect of these errors on SOVA. It was found that this decoder was less susceptible to these variations than Log-MAP.

The effect of Inter-Symbol Interference was then considered on these short frame codes. Turbo codes require both SNR estimates and fading amplitude estimates to effectively scale the information received, to facilitate the production of extrinsic information, which is what makes these codes so effective. The presence of ISI means that it is very hard to determine the channel fading amplitude for each received bit, making realistic transmission of turbo codes a problem.

Turbo equalisation has been used to counteract these effects, but also increases the complexity of the receiver. The use of decision feedback equalisers was therefore considered. These systems were combined with turbo codes and experiments were made with respect to appropriate channel scaling techniques as the data at the turbo decoder would already have been processed, reducing the use of the received information. Simulations were made over a variety of time-invariant and time-variant channels with mixed results.

The first channel to be considered was time-invariant and showed the harshest effect on uncoded data of all channels examined. Simulations were conducted using AWGN channel scaling and using no scaling at all. Results showed that turbo code, combined with two of the most popular DFE update algorithms exhibited performance that degraded, not only with each iteration, but also with each component decode. As the only change to the data used by each component decoder was to the extrinsic information passed between them, it is reasonable to assume that this had a detrimental effect on the results. It was found that the best results were obtained with a single component decoder and equalisation as results were not compromised by increasingly erroneous reliability information.

Of the three time-variant channels, the indoor office environment was the only situation in which the use of iterations improved results, showing that these systems could be used in circumstances with mild ISI and would show good performance. SOVA was still found to be the better performer at lower BERs, except in the vehicular situation. Under these circumstances, the superior decoding

abilities of the Log-MAP decoder and coupled with the fact that the decoder was less susceptible to fading amplitude errors, as found earlier, meant that this was the better decoder.

The use of AWGN channel scaling produced improved results over those where scaling was omitted, although it was found that, in certain circumstances, the degradation experienced with no scaling was minimal. It was also found that the lower complexity equaliser algorithm often produced better results.

In conclusion, this thesis has shown that not all conclusions drawn from investigation into long frame turbo codes are applicable to short frame turbo codes. To begin with, for systems that require good, but not necessarily optimal, bit error performance, with short frames and minimal delay, SOVA should be considered. Not only does it reach voice quality BERs at lower signal-to-noise ratios than Log-MAP decoding, it also does this with less complexity. It is also the case that, at low BERs, SOVA can outperform Log-MAP for equal complexity. Although SOVA is more susceptible to errors in channel amplitude estimation, it is more resilient to errors in signal-to-noise ratio estimation, both of which are required for scaling prior to turbo decoding.

In the case of channels exhibiting ISI, the combination of turbo codes with DFE should only be considered in cases where the ISI is mild, in which case AWGN channel scaling can be used to good effect and SOVA should again be considered. Where ISI is harsher, the DFE and single soft-decision decoding produces good results, but using these decoders in turbo configuration will not only induce a longer delay, it will also produce worse results.

7.2 Further work

It seems from the investigations made here that further investigation should be made into the SOVA turbo decoder. Not only has it been proved to be much less complex than the Log-MAP based scheme, but it has also proven itself able to compete with Log-MAP at bit error rates suitable to applications requiring the use of short frames.

The reduction in complexity coupled with the equal, or improved, bit error and frame error rates that this algorithm provides may well allow the turbo code scheme to again be considered for these applications in future mobile communications generations.

It has also been shown that, where the lowest possible bit error rates are of prime concern, it is possible to obtain improved bit error rates with SOVA over Log-MAP for an equal number of decoder operations by choosing a component code with higher minimum effective free distance for use with the SOVA decoder. The results and findings here serve only as an indication of the possibilities and further work is needed to substantiate them, with particular attention paid to the interleaver method used. As mentioned before, the error floors exhibited in some results are due to imperfect interleaver design and any improvements in this area may have an effect on further results. Because systems requiring very low bit error rates are likely to be less concerned with bandwidth puncturing may not be used either and

therefore attention should also be given to the effect that this has, as the SOVA algorithm has been shown to be more resilient to the loss of information than Log-MAP.

Ultimately, the combination of turbo codes with adaptive equalisation in the form of DFE was unsuccessful. It appears that it is the turbo codes inability to function with hard decisions, and without proper channel reliability scaling that is the combinations downfall. However, there is still a need for a low complexity solution. The turbo equalisation scheme would be more attractive if the overall complexity could be reduced, and, although the insertion of pilot symbols and channel equalisation cannot be avoided, the use of SOVA within the scheme, which is almost entirely researched with respect to the MAP based algorithms, may provide a suitable solution.

References

[ANG01] Ang W.P. and Garg H.K., *'A New Iterative Channel Estimator For The Log-MAP and Max-Log-MAP Turbo Decoder in Rayleigh Fading Channel'*, GLOBECOM '01, Vol.6, pp. 3252 -3256, November 2001.

[ANG02] Ang W.P. and Garg H.K., *'A New Adaptive Channel Estimator For Turbo Decoding In Rayleigh Fading Channels'*, ICCS 2002, pp. 26-28 November 2002

[BAH72] Bahl L. R., Cocke J., Jelinek F. and Raviv J., *'Optimal Decoding of Linear Codes for Minimising Symbol Error Rate'*, Abstracts of Papers, Int. Symp. Info. Theory, p. 90, January 1972

[BAH74] Bahl L. R., Cocke J., Jelinek F. and Raviv J., *'Optimal Decoding of Linear Codes for Minimising Symbol Error Rate'*, IEEE Trans. Info. Theory, Vol. IT-20, pp. 248-287, March 1974

[BAR94] Barbulescu A. S. and Pietrobon S. S., *'Interleaver Design for Turbo Codes'*, Electronics Letters, Vol. 30, No. 25, pp. 2107-2108, December 1994

[BAR95] Barbulescu A. S. and Pietrobon S. S., *'Terminating the trellis of Turbo Codes in the Same State'*, Electronics Letters, Vol. 31, No. 1, pp. 22-23, 1995

[BAT87] Battail G., *'Ponderation des symboles decodes par l'algorithem de Viterbi (In French)'*, Ann. Telecommun., Vol. 42, No. 1-2, pp. 31-38, January 1987

[BAU98] Bauch G. and Franz V.; *'Iterative equalization and decoding for the GSM-system'* Proc**.** 48th IEEE VTC 98, Vol. 3 , pp. 2262-2266, May 1998

[BEN96] Benedetto S. and Montorsi G., *'Unveiling Turbo-Codes: Some Results on Parallel Concatenated Coding Schemes'*, IEEE Trans. Info. Theory, Vol. 42, No. 2, pp. 409-428, March 1996

[BEN96a] Benedetto S. and Montorsi G., *'Design of Parallel Concatenated Convolutional Codes'*, IEEE Trans. Comms., Vol. 44, No. 5, pp. 591-600, May 1996

[BEN98] Benedetto S., Garello R. and Montorsi G., *'A Search for Good Convolutional Codes to be Used in the Construction of Turbo Codes'*, IEEE Trans. Comms., Vol. 46, No. 9, September 1998

[BER93] Berrou C., Glavieux A., Thitimajshima P., *'Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes(1)'*, Proc. IEEE Int. Conf. On Communications, pp. 1064-1070, May 1993

[BER93a] Berrou C., Adde P., Angui E. and Faudeil S., *'A Low Complexity Soft-Output Viterbi Decoder Architecture'*, Proc. ICC '93, pp. 737-740, May 1993

[BER93b] Berrou C. and Glavieux A., *'Turbo codes: General principles and applications'*, Proc. 6th Tirrenia Int. Workshop on Digital Comms., Sept. 1993

[BER96] Berrou C. and Jezequel M., *'Frame-Oriented Convolutional Turbo Codes'*, Electronics Letters, Vol. 32, No. 15, pp. 1362-1364, July 1996

[BER01] Berrou C., *'Turbo codes: some simple ideas for efficient communications'*, Proc. 7th Int. Workshop on DSP Techniques for Space Comms., Sesimbra, Portugal, Oct. 2001

[BER03] Berrou C., *'The Ten-Year-Old Turbo Codes are Entering into Service'*, IEEE Comms. Mag., pp. 110-116, August 2003

[BLA95] Blackert W. J., Hall E. K. and Wilson S. G., *'Turbo Code Termination and Interleaver Conditions'*, Electronics Letters, Vol. 31, No. 24, pp. 2082-2084, November 1995

[BRE99] Breiling M., Peeters S. and Huber J., *'Class of Double Terminating Turbo Code Interleavers'*, Electronics Letters, Vol. 35, No. 5, pp. 389-391, March 1999

[BUC70] Bucher, E. and Heller, J., *'Error probability bounds for systematic convolutional codes'*, IEEE trans. Info. Theory, Vol. 16, No. 2, pp. 219-224, March 1970

[BYU99] Byung G. L., Sang J. B., Seog G. K and Eon K. J., *'Design of Swap interleaver for Turbo Codes'*, Electronics Letters, Vol. 35, No. 22, pp. 1939-1940, October 1999

[CHA05] Chatzigeorgiou I., Rodrigues M. R. D., Wassell I. J. and Carrasco R., *'Punctured binary turbo-codes with optimized performance'*, Vehicular Tech. Conf. (VTC-Fall'05), September 2005

[DAI02] Dai H.; Poor V., *'Turbo multiuser detection for coded DMT VDSL systems'*, IEEE Journ. Select. Areas Comms., Vol. 20, No. 2, pp. 351-362, February 2002

[DAN99] Daneshgaran F. and Mondin M., *'Design of Interleavers for Turbo Codes: Iterative Interleaver Growth Algorithms for Polynomial Complexity'*, IEEE Trans. Info. Theory, Vol. 45, No. 6, pp. 1845-1859, 1999

[DEN00] Dennett C. and Carrasco R. A., *'Assessment of Turbo Codes over Fading Channels'*, CSNDSP'00, July 2000

[DEN04] Dennett C. and Carrasco R.A., *'A Comparison of Complexity and Performance of Turbo Codes over Rayleigh Fading Channels'*, CSNDSP'04, July 2004

[DIV95] Divsalar D., Pollara F., *'Turbo Codes for Deep-Space Communications'*, JPL TDA Progress Report, pp. 42-120, February 1995

[DIV95a] Divsalar D. and Pollara F., *'Turbo Codes for PCS Applications'*, Proc. ICC '95, pp. 54-59, June 1995

[DIV96] Divsalar D. and McEliece R. J., *'The Effective Free Distance of Turbo Codes'*, Electronics Letters, Vol. 32, No. 5, pp. 445-446, February 1996

[ERF94] Erfanian J. A., Pasupathy S. and Gulot G., *'Reduced Complexity Symbol Detectors with Parallel Structures for ISI Channels'*, IEEE Trans. Comm., Vol. 42, pp. 1661-1671, February 1994

[FOR70] Forney G. D. Jnr., *'Convolutional Codes I: Algebraic Structure'*, IEEE Trans. Info. Theory. Vol. IT-16, no. 16, pp. 720-738, November 1970

[FOR72] Forney G. D. Jnr., *'Maximum-Likelihood Sequence Estimator of Digital Sequences in the Presence of Intersymbol Interference'*, IEEE Trans. Info. Theory, Vol. IT-18, No. 3, pp. 363 – 378, May 1972.

[FOR73] Forney G. D. Jnr., *'The Viterbi Algorithm'*, Proc. IEEE vol. 61, pp. 268-278, March 1973

[GEO71] George D.A., Bowen R.R. and Storey J.R., *'An Adaptive Decision Feedback Equaliser'*, IEEE Trans. Comm. Tech., Vol. COM-19, pp. 281 – 293, Jun 1971.

[GER69] Gersho A., *'Adaptive Equalisation of Highly Dispersive Channels'*, Bell Syst. Tech. Journal, Vol. 48, pp. 55 – 70, Jan 1969.

[GIT92] Gitlin R. D., Hayes J. F., Weinstein S.B., *'Data Communications Principles'*, New York: Plenum, 1992

[HAG89] Hagenauer J. and Hoeher P., *'A Viterbi Algorithm with Soft-Decision Outputs and its Applications'*, Proc. Globecom'89, pp. 47.11-47.17, November 1989

[HAG95] Hagenauer J. and Robertson P., *'Iterative ("Turbo") Decoding of Systematic Convolutional Codes with the MAP and SOVA Algorithms'*, ITG-Fachberichte, Vol. 130, pp. 21-29, 1995

[HAG96] Hagenauer J., Offer E. and Papke L., *'Iterative Decoding of Binary Block and Convolutional Codes'*, IEEE Trans. Inf. Theory, Vol. 42, No. 2, March 1996

[HAG01] Hagenauer J. and Bauer R., *'The turbo principle in joint source channel decoding of variable length codes'*, IEEE Proc. Inf. Theory Workshop, pp. 33-35, September 2001

[HAG03] Hagenauer J and Gortz N., '*The turbo principle in joint source-channel coding'*, Proc. IEEE Inf. Theory. Workshop, pp. 275-278, March 2003

[HAL98] Hall E. K. and Wilson S. G., *'Design and Analysis of Turbo Codes on Rayleigh Fading Channels'*, IEEE Journ. Select. Areas Comms, Vol. 16, No. 2, February 1998

[HAY91] Haykin S., '*Adaptive Filter Theory'*, 2nd Edition, Prentice-Hall, 1991

[HAY01] Haykin S., '*Communication Systems'*, 4th Edition, Wiley, 2001

[HMI99] Hmimy A. H. and Gupta S. C., *'Performance of Turbo-Codes for WCDMA Systems in Frequency Selective Fading Channels'*, IEEE 49th Vehicular Technology Conference, Vol. 2, pp. 1459-1463, July 1999

[HO98] Ho M. S. C., Pietrobon S. S. and Giles T., *'Optimising the constituent codes for turbo decoders'*, IEEE Int. Symp. Info. Theory, p. 178, August 1998

[HO98a] Ho M. S. C., Pietrobon S. S. and Giles T., *'Improving the constituent codes of turbo encoders'*, IEEE Global Telecomms. Conf., vol. 6, pp. 3525-3529, November 1998

[HO98b] Ho M. S. C., Pietrobon S. S. and Giles T., *'Interleavers for punctured turbo codes'*, IEEE Asia-Pacific Conf. on Comms. and Singapore Int. Conf. on Comms. Systems, vol. 2, pp. 520-524, November 1998.

[HOS00] Hoshyar R., Jamali S. H. and Locus C., *'Ant Colony Algorithm for Finding good Interleaving Pattern in Turbo Codes'*, IEE Proc. Comms., Vol. 147, No. 5, pp. 257-262, October 2000

[HUB90] Hubing N.E. and Alexander S.T., *'Statistical Analysis of Soft Constrained Initialisation of Recursive Least Squares Algorithm'*, Proc. ICASSP, Albuquerque New Mexico, 1990.

[JAK74] Jakes W. C., *'Microwave Mobile Communications'*, Bell Telephone Laboratories, Wiley and Sons, 1974

[JER84] Jeruchim M. C., *'Techniques for estimating Bit Error Rate in the Simulation of Digital Communications Systems'*, IEEE Journ. Select. Areas Comms., Vol. 2, No. 1, pp. 153-170, January 1984

[JOE94] Joerssoen O. and Meyr H., *'Terminating the Trellis of Turbo-Codes'*, Electronic Letters, Vol. 30, No. 16, pp.1285-1286, August 1994

[JUN94] Jung P. and Nasshan M., *'Dependence of the Error Performance of Turbo-Codes on the Interleaver Structure in Short Frame Transmission Systems'*, Electronics Letters, Vol. 30, No. 4, pp. 287-288, February 1994

[JUN94a] Jung P. and Nasshan M., *'Performance Evaluation of Turbo Codes for Short Frame Transmission Systems'*, Electronics Letters, Vol. 30, No. 2, pp. 111-113, January 1994

[JUN96] Jung P., *'Comparison of Turbo-Code Decoders Applied to Short Frame Transmission Systems'*, IEEE Journ. Select. Areas Comms., Vol. 14, No. 3, pp. 530-537, April 1996

[KAZ99] Kazi S. and Lucke L., *'Adaptive LMS filter receiver for a turbo coded CDMA system'*, RAWCON99, pp. 69-72, August 1999

[KOC90] Kock W. and Baier A., *'Optimum and sub-Optimum Detection of Coded Data Disturbed by Time-Varying Intersymbol Interference'*, Proc. GLOBECOM '90, pp. 1679-1684, December 1990

[KOO97] Koorapaty H., Wang Y., and Balachandran K., *'Performance of turbo codes with short frame sizes'*, Proc. IEEE Veh. Tech. Conf., pp. 329-33, 1997.

[LIN97] Lin L. and Cheng R. S., *'Improvements in SOVA-Based Decoding For Turbo Codes'*, Proc. IEEE Int. Conf. Comms, pp. 1473-1478, June 1997

[LIU04] Liu L. and Ping L., *'An Extending Window MMSE Turbo Equalization Algorithm'*, IEEE Sig. Process. Letters, Vol. 11, No. 11, pp. 891-894, November 2004

[LUC66] Lucky R.W., *'Techniques for Adaptive Equalisation of Digital Communication Systems'*, Bell Syst. Tech. Journal, Vol. 45, pp. 255 – 286, Feb 1966.

[LUC68] Lucky R.W., Salz J., *'Principles of Data Communication'*, McGraw-Hill, 1968.

[McA72] McAdam P. L., Welch L. and Weber C., *' M.A.P. Bit Decoding of Convolutional Codes'*, Abstracts of Papers, Int. Symp. Info. Theory, p. 91, January 1972

[MUE81] Mueller M.S., *'Least-Squares Algorithms for Adaptive Equalisers'*, Bell Syst. Tech. Journal, Vol. 60, pp. 1905 – 1925, Oct 1981.

[PAP96] Papke L. and Robertson P., *'Improved Decoding with the SOVA in a Parallel Concatenated (Turbo-Code) Scheme'*, Proc. ICC '96, pp. 102-106, July 1996

[PAR00] Parsons J. D., *'The Mobile Radio Propagation Channel, 2nd edition '*, Wiley and sons, 2000

[PER96] Perez L. C., Seghers J. and Costello D. J., *'A Distance Spectrum Interpretation of Turbo Codes'*, IEEE Trans. Info. Theory, Vol. 42, No. 6, pp. 1698-1709, November 1996

[PRO75] Proakis J.G., *'Advances in Equalisation for Intersymbol Interference'*, Advances in Communication Systems, Vol. 4, Advance, 1975

[PRO95] Proakis, J.G., *'Digital Communications'*, 3rd Edition, McGraw-Hill, 1995

[ROB94] Robertson P., *'Illuminating the Structure of Parallel Concatenated Recursive Systematic (TURBO) Codes'*, Proc. GLOBECOM '94, pp. 1298-1303, November 1994

[ROB95] Robertson, P., Villebrun, E. and Hoeher, P., *'A Comparison of Optimal and Sub-optimal MAP decoding Algorithms Operating in the Log Domain'*, Proc. ICC'95, June 1995

[RYA99] Ryan, W. E., *'A Turbo Code Tutorial'*, Unpublished Paper available at www.eccpage.com/turbo2c.ps

[SAD00] Sadjadpour H. R., Sloane N. J. A., Salehi M. and Nebe G., *'Interleaver Design for Turbo Codes'*, IEEE Select. Areas Comms, Vol. 19, No. 5, pp. 831-837, May 2001

[SAI99] Said F., Aghvami A. H. and Chambers W. E., *'Improving Random Interleaver for Turbo Codes'*, Electronics Letters, Vol. 35, No. 25, pp.2194-2195, December 1999

[SAL73] Salz J., *'Optimum Mean-Square Decision Feedback Equalisation'*, Bell Syst. Tech. Journal, Vol. 52, pp. 1341 – 1373, Oct 1973.

[SHA48] Shannon C. E., *'A Mathematical Theory of Communication'*, Bell Syst. Tech. Journal, Vol. 27, pp. 379-423 (July) and pp. 623-656 (October)

[SKL97] Sklar B., *'Rayleigh Fading Channels in Mobile Digital Communication Systems Part 1: Characterisation'*, IEEE Comms. Mag., pp. 90-100, July 1997

[TAN99] Tang K., Siegel P. H. and Milstein L. B., *'On the performance of turbo coding for the land mobile channel with delay constraints'*, Proceedings Asilomar Conference on Signals, Systems & Computers, pp. 1659-1664, October 1999.

[TUC02] Tüchler M., Singer A. C. and Koetter R., *'Minimum Mean Squared Error Equalization Using A Priori Information'*, IEEE Trans. Sig. Proc., Vol. 50, No. 3, pp. 673-683, March 2002

[UMTS1] Universal Mobile Telecommunications Systems (UMTS) *'Selection procedures for the choice of radio transmission technologies for the UMTS (UMTS 30.03 version 3.2.0)'*, ETSI, April 1998

[UNG72] Ungerboeck G., *'Theory on the Speed Convergence in Adaptive Equalisers for Digital Communications'*, IBM Journ. Res. Dev, Vol. 16, pp. 546 – 555, 1972.

[VAL98] Valenti M.C. and Woerner B.D., *'Performance of turbo codes in interleaved flat fading channels with estimated channel state information'*, IEEE VTC 98, Vol. 1, pp. 66-70, May 1998

[VAL01] Valenti M. C.; Woerner B. D., *'Iterative multiuser detection, macrodiversity combining, and decoding for the TDMA cellular uplink'*, IEEE Journ. Select. Areas in Comms., Vol. 19 , No. 8 , pp. 1570-1583, August 2001

[VAL01a] Valenti M. C.; Woerner B. D., *'Iterative Channel Estimation and Decoding of Pilot Symbol Assisted Turbo Codes Over Flat-Fading Channels'*, IEEE Journ. Select. Areas in Comms., Vol. 19, No. 9, pp. 1697-1705, September 2001

[VAL01b] Valenti M. C. and Sun J., *'The UMTS Turbo Code and an Efficient Decoder Implementation Suitable for Software-Defined Radios'*, Int. Jour. Wireless Info. Networks, Vol. 8, No. 4, pp. 203-215, October 2001

[VAT02] Vatta F., Montorsi G. and Babich F., *'Analysis and Simulation of Turbo Codes Performance over Rice Fading Channels'*, ICC2002, Vol. 25, No. 1, pp. 1506-1510, April 2002

[VIT67] Viterbi, A. J., *'Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm'*, IEEE Trans. Info. Theory, vol. IT-13, pp. 260-269, April 1967

[VIT71] Viterbi A. J., *'Convolutional Codes and Their Performance in Communications Systems'*, IEEE Trans. Comms. Tech., Vol. COM-19, No. 15, pp. 751-771, October 1971

[VUC00] Vucetic B. and Yuan J., *'Turbo Codes: Principles and Applications'*, Kluwer Academic Publishers, 2000

[WID60] Widrow B., Hoff M.E.Jnr., *'Adaptive Switching Circuits'*, Inst. Radio Engineers, Western Electronic Show and Convention, Convention Record, Pt. 4, pp. 96 – 104, Aug 1960.

[WID70] Widrow B., *'Adaptive Filters'*, Holt, Rinehart and Winston, 1970.

[WOO00] Woodard, J. P., Hanzo, L. *'Comparative Study of Turbo Decoding Techniques: An Overview'* IEEE Transactions on Vehicular Technology, Vol. 49, No. 6, pp. 2208-2233, November 2000

[WOR00] Worm A, Hoeher P. and When N., *'Turbo Decoding Without SNR Estimation'*, IEEE Comms. Letters, Vol. 4, No. 6, pp. 193-193, June 2000

[YUA99] Yuan J., Vucetic B. and Wen F., *'Combined Turbo Codes and Interleaver Design'*, IEEE Trans. Comms. Vol. 47, No. 4, pp. 484-487, April 1999

Appendix A

Redundancy

To better understand the abilities and structure of turbo codes as described in chapter 4, it is necessary to first understand the basic features of an error control coding scheme. This appendix and those following show the concepts of error control coding and the conventional coding schemes from which turbo codes developed. The most basic concept of error control coding is that of redundancy.

Assume that the following set of datawords are equally likely to be transmitted:

00	01
10	11

Table A.1: Example Dataword Set

If no coding strategy were present, the dataword ‘00’ would be transmitted unchanged. Thus, when the dataword ‘00’ is received, there is no way of knowing whether an error has been induced during transmission. As all possible two-bit, binary words are included in the dataword set and are equally likely, the received ‘00’ could be uncorrupted, have suffered slight noise, or have been completely changed. To avoid this type of situation, extra binary digits (bits), referred to as parity bits, can be added to each dataword, by way of an example:

000	011
101	110

Table A.2: Example Codeword Set

Each one of the codewords in the above set corresponds to one of the datawords of table A.1. Unlike the dataword set of table A.1, this codeword set contains only half of the total possible three-bit words (Alphabet):

000	100
001	101
010	110
011	111

Table A.3: Three-bit Word Alphabet

The full three-bit word alphabet has four words that are unused by the code and are therefore referred to as redundant. Thus a change in a codeword occurring during transmission will not necessarily produce another codeword, introducing an element of reliability to any decisions made about whether to trust the received word. In fact, if, during transmission, one bit of the codeword is altered, the receiver will know that an error has occurred. In effect what has happened by adding an extra bit to the codeword is that the possible words have become more different from one another.
It should be noted that redundancy is only useful if the method in which it was applied is controlled. If there is no method to the addition of the redundant (or parity) bits, then it is not possible to interpret them at the receiver. Error control codes add redundancy to data in a controlled fashion, allowing the detection and possible correction of errors occurring during transmission.

The addition of parity bits to datawords, while increasing the error control capabilities of the code, also reduces the rate of the code. This is a measure of the code efficiency and is defined as:

$$Rate\ R = \frac{Dataword\ Bits\ at\ time\ t}{Codeword\ bits\ at\ time\ t}$$

Applying this formula to the original dataword set (table A.1) and the codeword set (table A.2), it is clear that the addition of the parity bit has reduced the rate of the code considerably:

$$\frac{2}{2} = 1 \ = 100\% \text{ efficiency for the dataword set}$$

And

$$\frac{2}{3} = 0.666 \ = 66.6\% \text{ efficiency for the equivalent code with one parity bit.}$$

This simply shows that, of all the bits received after transmission, 100% of them represent data information for the uncoded two-bit dataword example of table A.1, whereas for the basic coded data of table A.2, only 66% of what is received is actually useful information.

Appendix B

The Error Control Capabilities of a Code

By adding parity bits to a dataword and thus creating a codeword as described in appendix A, the designer is effectively reducing the number of acceptable codewords from the set of all possible words. In the example of appendix A, once the third bit has been added, there are four possible codewords. There are, however, a total of eight available binary words with three bits:

000	100
001	**101**
010	**110**
011	111

Table B.1: Example Codewords as Part of Their Alphabet

Dataword	Codeword
00	00011
01	01000
10	10110
11	11101

Table B.2: Five-bit Codewords

Whereas with the original four datawords, a single error would transform one dataword into another, with the four codewords it would take at least two errors to convert one into another. This serves to illustrate the fact that error detection and, perhaps, correction is possible because some of the words of the alphabet are not used. Basically, the more unused words available, the better the error detection and correction capabilities of the code, although it is not quite that simple. The flip side of this is that the code rate is reduced. For instance, applying three parity bits to the datawords of table

A.1, to produce table B.2 above, gives a minimum of three errors necessary to transform one codeword to another, however, the rate is reduced to 0.4, or 40% efficiency.

Appendix C

Modulo-*q* Arithmetic

To introduce controlled redundancy, error control codes make use of modulo-*q* arithmetic, where *q* represents the field of values that a particular element in a codeword can take. These fields are referred to as Galois Fields, denoted GF(*q*), and can only be constructed if *q* is prime or a multiple of a prime.

All Galois Fields must contain a null (0) element and a unary (1) element, implying that the simplest field is binary or GF(2). As with the fields of real and complex numbers, the operations performed on Galois fields must adhere to set rules. However, a Galois field has a finite number of elements, unlike its real and complex counterparts. This means that the conventional rules for addition and multiplication are redundant. For these operations on finite fields, modulo-*q* arithmetic is employed.

Table C.1 shows the operations and the rules of modulo-*q* addition and multiplication according to Proakis [PRO95] are given below:

Rules of Modulo-*q* addition

a. The set of possible values (F) is closed under addition, i.e., if $a,b \in F$, then $a+b \in F$.

b. Addition is associative. Which means that if a, b and c are all elements of the set F, then $a+(b+c) = (a+b)+c$.

c. Addition is commutative. In other words $a+b = b+a$.

d. An element of the set is zero, which satisfies the condition $a + 0 = a$.

e. Each element in the set has a corresponding negative element. Therefore, b would have its negative denoted by $-b$. The subtraction of two elements, for instance $a-b$ is defined as $a + (-b)$.

Rules of Modulo-*q* multiplication

a. The set F is closed under multiplication, so if $a,b \in F$, then $ab \in F$.

b. Multiplication is associative. Hence $a(bc) = (ab)c$.

c. Multiplication is commutative, i.e., $ab = ba$.

d. Multiplication is distributive over addition. In other words, $(a+b)c$ can also be represented as $ac+bc$.

e. Set F contains an element called the identity which satisfies the condition $a(1) = a$, for any element of F.

f. All elements in the set F except zero have an inverse. Consequently, if $b \in F (b \neq 0)$ then it's inverse is b^{-1} and $bb^{-1} = 1$. Division of two elements, for example a/b, is defined as ab^{-1}.

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Table C.1: Modulo-2 Arithmetic

Appendix D

The Minimum Hamming Distance of a Code

Appendix C showed how parity bits are added to datawords in a controlled manner. Appendix D looks at the minimum Hamming distance of a code, which is a factor used to determine the error control capabilities of that particular code.

Distance, in error control coding terms, is the number of positions in which two words differ. For instance, the word '00' differs in one position from the word '01' and two positions from the word '11', therefore the respective distances of the codewords from '00' are 1 and two.

In its most general form, the minimum Hamming distance, d_{min}, is defined as the lowest number of positions that any two unlike codewords differ. Or:

$$d_{min} = min\{w(x_i \oplus x_j)\} \quad i \neq j \tag{D.1}$$

Where x_i and x_j are codewords and $w(x)$ represents the number of non-zero elements in the codeword and is referred to as the weight of the codeword. The symbol '⊕' signifies the exclusive-or operation. Note that if the code is linear, one codeword summed with another under modulo-q addition results in another codeword, simplifying equation (D.1) to:

$$d_{min} = min\{w(x_i)\} \qquad i \neq 0 \tag{D.2}$$

Where x_i is any codeword other than the all-zero codeword.

The minimum Hamming distance can also be defined as:

$$d_{\min} = e + t + 1 \tag{D.3}$$

Where e is the number of detectable errors and t is the number of correctable errors. Obviously, it is not possible to correct an error unless it has been detected, so $e \geq t$. Equation (D.3) can be transposed to give the maximum error correcting capabilities of a particular code by setting $t = e$, leading to:

$$t = \frac{d - 1}{2} \tag{D.4}$$

Appendix E

Convolutional Codes

Convolutional codes are one of the most common types of error control codes, using simple encoding methods and achieving high performance due to their virtually infinite codeword length and highly developed decoding techniques.

The basic building block of the convolutional encoder is the linear finite-state shift register. The binary convolutional encoder is comprised of a group of these registers. Taps leading from each register have a gain of zero or one, indicating an open circuit or short circuit respectively, and are combined using modulo-2 addition. The sum of these taps is therefore a combination of the present data bit and a number of preceding data bits (referred to as the state of the register). This implies that a data bit will have an effect on more than one code bit.

The same bank of registers is tapped more than once to provide the extra bits necessary for error control.

Figure E.1 gives an example of a convolutional encoder.

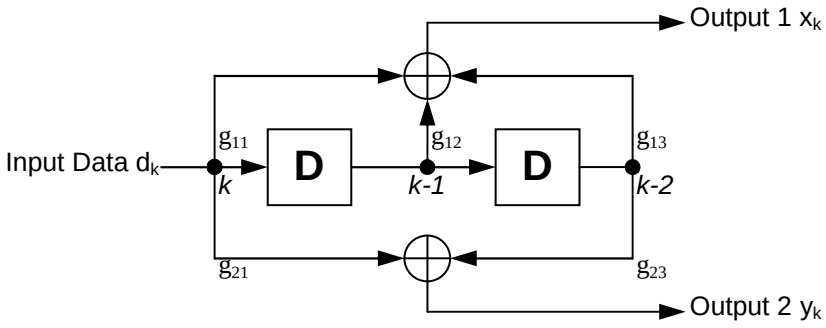


Figure E.1: Non-Systematic Convolutional Encoder

Assume that the encoder is binary. Input data is passed through the *m* shift registers, *k* is the current information bit, *k-1* is the previous information bit and *k-2* the information bit at time *t-2*.

Variable *m* is the memory length and is determined by the number of shift registers in the encoder, two in this example. The first output is the modulo-2 sum of all three data bits, whereas the second output is the sum of the current data bit and the data bit at time *t-2*. The coding rate *R* is given by the number of inputs (*k*) over the number of outputs (*n*) at any point in time, in the case of figure E.1 the rate is 1/2.

The constraint length *K* represents the number of shifts a single bit can make before it no longer affects the output of the encoder. *K* is defined by:

$$K = m + 1 \tag{E.1}$$

The number of states (*S*) is determined by the equation:

$$S = q^m \tag{E.2}$$

Where *q* is the index of the Galois field. Thus, for this binary example, the number of memory elements is two, the index of the Galois field is two and the number of possible memory states is 4.

To represent the code numerically, each of the *n* modulo-*q* adders is specified with a $k \times K$ vector representing the connections of the encoder to that adder. A binary high (1) is used to indicate a connection, although with some non-binary convolutional encoders, an integer value from within the Galois field may be used as a multiplier before the adder, in this case the connection would be indicated by that value. Returning to the example, the dimensions of the vectors are $k = 1$ and $K = 3$, and the vector representing the first output is $g_1 = [g_{11},g_{12},g_{13}] = [111]$. The vector representing output two is $g_2 = [g_{21},g_{22},g_{23}] = [101]$.

It is usual to combine these vectors to produce a generator matrix and for the code to become known as a $[111;101]_2$ convolutional encoder, or in its octal form as $[7;5]_8$ for brevity.

The example encoder of figure E.1 is of a non-systematic convolutional encoder. The term systematic is used to indicate that the input bit is included in the output of the encoder. This implies that a systematic encoder will have one less modulo-*q* adder in the system than a non-systematic encoder of the same rate and constraint length. For example, figure E.2 gives an example of a systematic encoder with similar rate and constraint length as the original example encoder.

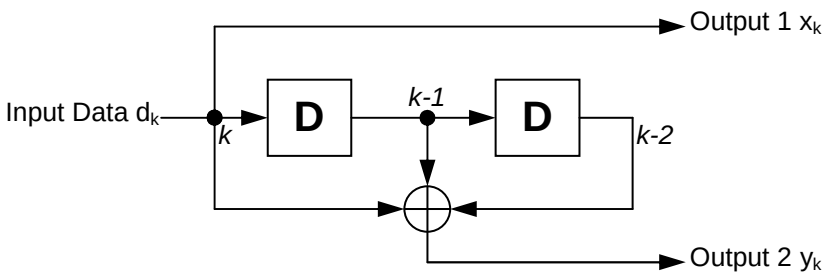


Figure E.2: Systematic Convolutional Encoder, $R = 1/2$, $K = 3$

Appendix F

Representing the Output of Convolutional Codes and Finding the Hamming Distance

Appendix F shows how to determine the Hamming distance of a convolutional code, the methods of representation of a codes output also help to explain the decoder process later.

The simplest method used to represent the output of a convolutional encoder, the code tree, expands horizontally with time. The branches spread out from the centre to show all possible codeword outputs. An example of a code tree for the non-systematic $[7;5]_8$ example above is given in figure F.1. As with all representations, it is assumed that the registers begin at the all-zero state, that is, all registers are assumed to contain zero values.

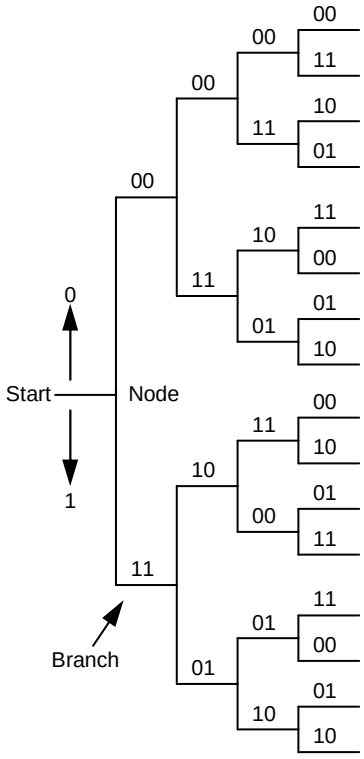


Figure F.1: Code Tree Diagram for Non-Systematic $[7;5]_8$ Convolutional Code

The labels at each branch are the output bits corresponding to that particular branch. The convention is that the upper branch leaving any node represents a binary ‘0’ and that the lower branch corresponds to a binary ‘1’. The tree grows exponentially with each added input, which relates to each step in time, and soon becomes cumbersome to work with. Observe that the tree repeats itself after the third set of branches. As a result of this repetition, the matching nodes can be united. This reducing process leads to the second method used to illustrate the output of a convolutional encoder, the trellis diagram.

The trellis diagram is a collapsed version of the code tree. Referred to as such because of its remerging branches. Figure F.2 below illustrates this method.

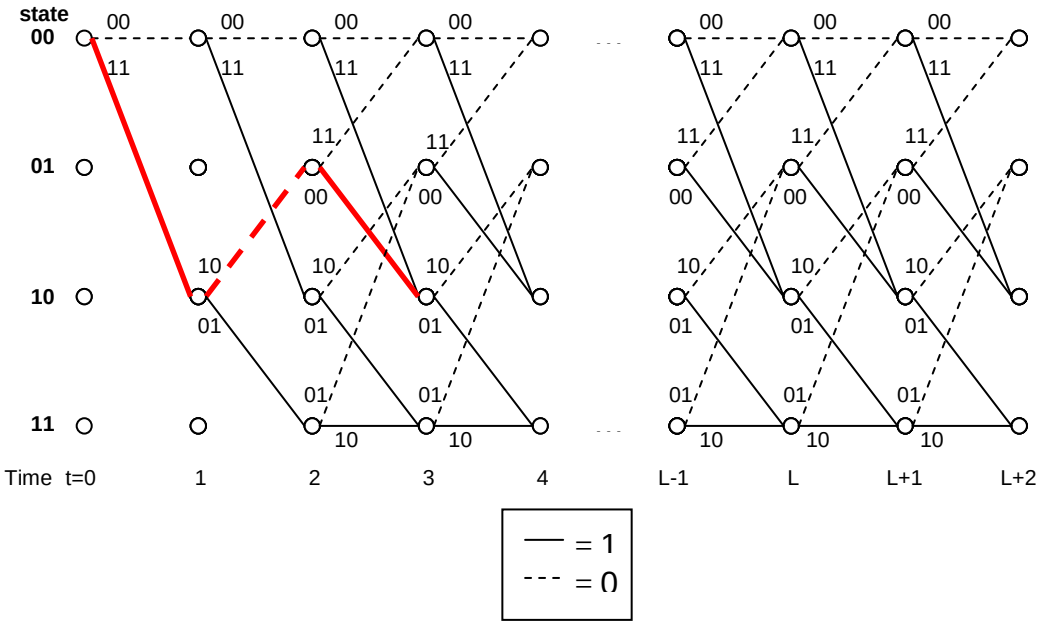


Figure F.2: Trellis Diagram for Non-Systematic $[7;5]_8$ Convolutional Code

The trellis, like the code-tree, extends horizontally with time *t* and the vertical states represent the four possible combinations of bits stored in the memory elements. The branch labels indicate the encoder output and the key describes the input values. As an example, assume that the encoder of figure E.1 is initialised (all registers set to zero). After receiving a binary '1' followed by a binary '0', the next data bit is another '1'. This would mean that the memory state at time $t = 2$, after the first two input bits, would be '01'. The output of the encoder at the transition triggered by the third bit would be '00' and the shift registers would shift out the '1' in the second memory position, replacing it with the '0' previously in first position, which itself would be replaced with the input bit '1'. These first three transitions have been highlighted in figure F.2.

As discussed in appendix D, the minimum Hamming distance is used to indicate the error control capabilities of a code. The method for finding d_{min}, described in that section still holds for convolutional codes, however, as the number of possible code words is semi-infinite, it is not practical. Here the trellis or state diagram proves useful. Considering the trellis, the lowest weight of all possible paths that merge with the all-zero path at any time is equal to the minimum Hamming distance. Using the trellis as an example, it can be seen in figure F.3 that at time $t = 3$, the lowest weight of all possible paths merging with the all-zero path is five, therefore, $d_{min} = 5$.

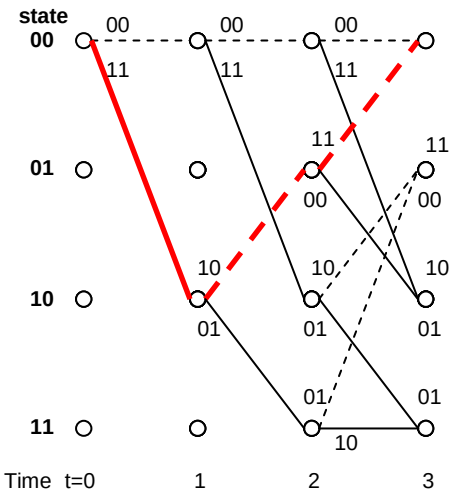


Figure F.3: Example Trellis Diagram Showing Minimum Hamming Distance

Note that the path with the lowest weight diverged from the all-zero path three time steps earlier, this would be the same at any point in time. Also notice that input bits corresponding to that path are always 000…100, differing with the all-zero path in only one position.

The third method used to represent convolutional codes is the state transition diagram, an example of which is given in figure F.4. This method is similar to the trellis diagram, but does not extend with time.

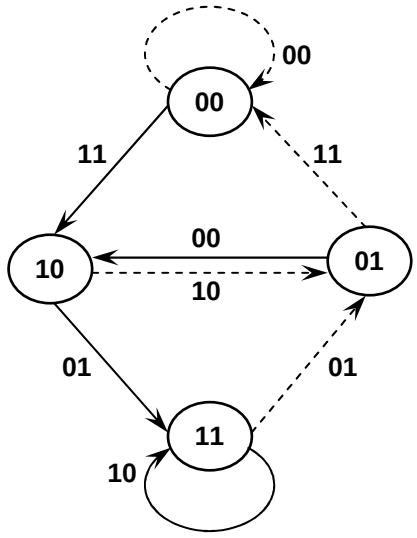


Figure F.4: State Transition Diagram for Non-Systematic $[7;5]_8$ Convolutional Encoder

As mentioned earlier, the bits stored in memory are referred to as the current state of the memory. The trellis diagram shows that the output of the encoder is a function of the current input bit

and the state. This means that if a view of the code structure is not necessary with reference to the time domain, then the trellis can be further simplified to become the state transition diagram.

The state transition diagram can also be used to return a closed form expression, whose expansion yields all the distance information. The diagram must first be redrawn thus:

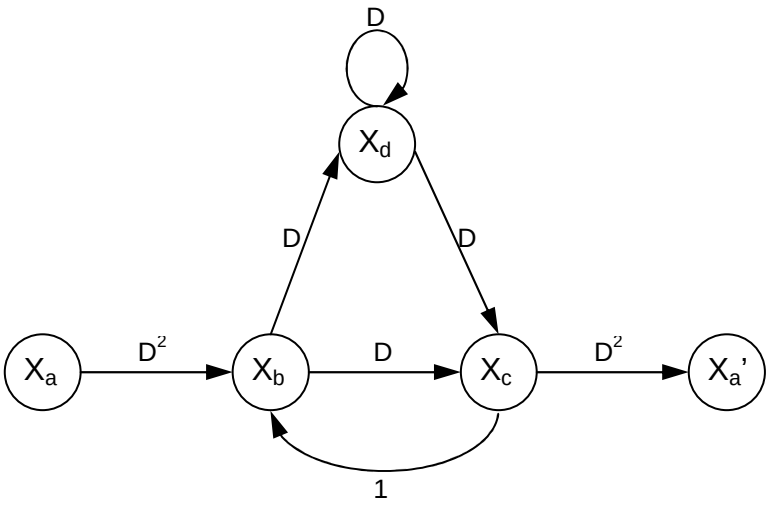


Figure F.5: State Transition Diagram with Weights

Here X_a and X_a' represent memory state 00, X_b represents state 10, X_c represents 01 and X_d is state 11. State 00 has been split and the circulating loop has been ignored as this corresponds to the all-zero codeword. Each branch is labelled $D^{weight\ of\ output}$, therefore a branch output of 00 becomes $D^0 = 1$, 01 and 10 become D^1 and so on. It is now possible to obtain the generating function of all paths merging with the all-zero path if the new diagram is regarded as a signal flow graph and the transfer function of that graph. Continuing the example:

$$X_b = D^2 X_a + X_c$$

$$X_c = DX_b + DX_d$$

$$X_d = DX_b + DX_d$$

$$X_a^{'} = D^2 X_c$$

$$X_c = X_d$$

$$\therefore X_c = DX_b + DX_c$$

$$\frac{(1-D)}{D} X_c = X_b$$

$$\therefore \frac{(1-D)}{D} X_c = D^2 X_a + X_c$$

$$\therefore X_a = \frac{(1-2D)}{D^3} X_c$$

$$\frac{X_a^{'}}{X_a} = \frac{D^5}{1-2D} = D^5 + 2D^6 + 4D^7 + ... + 2^k D^{k+5}$$

Giving a single path that merges with the all-zero path with a d_{min} of 5.

The performance of a code is dependent on the relative distances between code words, as explained above. Viterbi [VIT67] points out that the removal of an adder, as with the systematic form of the convolutional encoder (figure E.2) results in a reduction in d_{min}. Figure F.6 shows the trellis for the systematic encoder of figure E.2, with a path corresponding to d_{min} at time step 3 highlighted.

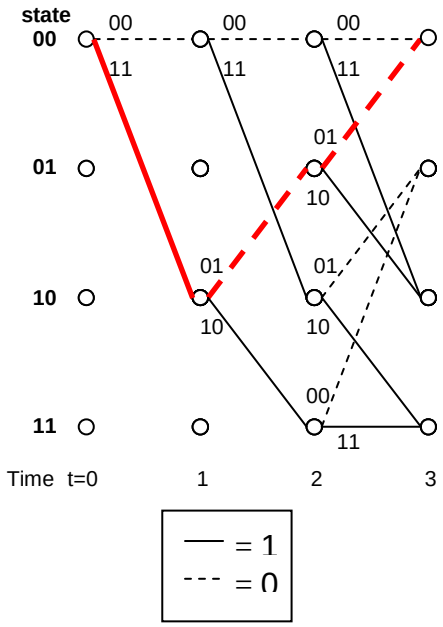


Figure F.6: Trellis Representation of Systematic Encoder of Figure E.2

It is evident from this diagram that the systematic convolutional encoder, although having equal constraint length *K* and rate *R*, has a d_{min} of 4, one less than the non-systematic encoder of figure E.1. Bucher and Heller [BUC70] have shown that for large constraint length codes, the performance of a systematic code is approximately equal to a non-systematic code of constraint length $K_{non\text{-}sys} = K_{sys}(1-R)$.

Appendix G

Matrix Inversion Lemma

MIL states that given two positive-definite matrices *A* and *B* both of dimensions *N by N*, following the relation:

$$A = B^{-1} + CD^{-1}C^{H} \tag{G.1}$$

Where ^{H} is Hermitian transposition (combined transposition and complex conjugation), *D* is a positive-definite matrix of dimensions *M by M* and *C* is an *N by M* matrix.

The inverse of *A* can be written thus:

$$A^{-1} = B - BC(D + C^{H}BC)^{-1}C^{H}B \tag{G.2}$$

As proof, the product of a matrix and its inverse must equal the identity matrix *I*. Applying this to equations (L.1) and (L.2) returns:

$$\begin{aligned}&\left[B^{-1} + CD^{-1}C^{H}\right]\left[B - BC(D + C^{H}BC)^{-1}C^{H}B\right]\\&= B^{-1}B - B^{-1}BC(D + C^{H}BC)^{-1}C^{H}B + CD^{-1}C^{H}B - CD^{-1}C^{H}BC(D + C^{H}BC)^{-1}C^{H}B\end{aligned} \tag{G.3}$$

$$BB^{-1} = I \tag{G.4}$$

$$= I - IC(D + C^{H}BC)^{-1}C^{H}B + CD^{-1}C^{H}B - CD^{-1}C^{H}BC(D + C^{H}BC)^{-1}C^{H}B \tag{G.5}$$

$$I \times C = C \tag{G.6}$$

Multiplying the third factor of equation (L.5) by $(D+C^H BC)(D+C^H BC)^{-1}$ returns:

$$\begin{aligned} &= I - C(D+C^H BC)^{-1}C^H B + CD^{-1}(D+C^H BC)(D+C^H BC)^{-1}C^H B \\ &-CD^{-1}C^H BC(D+C^H BC)^{-1}C^H B \end{aligned} \tag{G.7}$$

Expanding the third factor:

$$\begin{aligned} &= I - C(D+C^H BC)^{-1}C^H B + CD^{-1}D(D+C^H BC)^{-1}C^H B + CD^{-1}C^H BC(D+C^H BC)^{-1}C^H B \\ &-CD^{-1}C^H BC(D+C^H BC)^{-1}C^H B \end{aligned} \tag{G.8}$$

Substituting $DD^{-1} = I'$ to differentiate between *N by N* dimension identity matrices from *M by M* dimension matrices leaves:

$$= I - C(D+C^H BC)^{-1}C^H B + CI'(D+C^H BC)^{-1}C^H B \tag{G.9}$$

CI' can be replaced by *C*, therefore

$$= I - C(D+C^H BC)^{-1}C^H B + C(D+C^H BC)^{-1}C^H B = I \tag{G.10}$$

Therefore proving the Matrix Inversion Lemma.